GEORGIA INSTITUTE OF TECHNOLOGY LIBRARY

Regulations for the Use of Theses

Unpublished theses submitted for the Master's and Doctor's degrees and deposited in the Georgia Institute of Technology Library are open for inspection and consultation, but must be used with due regard for the rights of the authors. Passages may be copied only with permission of the authors, and proper credit must be given in subsequent written or published work. Extensive copying or publication of the thesis in whole or in part requires the consent of the Dean of the Graduate Division of the Georgia Institute of Technology.

This thesis by Ronaldo Gomes Ferraz has been used by the following persons, whose signatures attest their acceptance of the above restrictions.

A library which borrows this thesis for use by its patrons is expected to secure the signature of each user.

NAME AND ADDRESS OF USER	BORROWING LIBRARY	DATE

CONTINUITY CONSIDERATIONS IN CYCLIC PROJECT SCHEDULING

A THESIS

Presented to

The Faculty of the Division of Graduate

Studies and Research

by

Ronaldo Gomes Ferraz

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Industrial Engineering

Georgia Institute of Technology

March, 1973

CONTINUITY CONSIDERATIONS IN CYCLIC PROJECT SCHEDULING

Approved:

J. Gordon Davis, Chairman

Norman R. Baker

Richard H. Deane

Date approved by Chairman: May 7, 1973

ACKNOWLEDGMENTS

I wish to express my deepest appreciation to my advisor, Dr. J. Gordon Davis, for his many hours of advice and assistance without which this Thesis would never have been completed.

I am also grateful to the members of my reading committee, Drs. Norman R. Baker and Richard H. Deane, for their constructive criticism and suggestions which proved so helpful in producing the final form of the manuscript.

Finally, I wish to thank my wife, Vera, for her understanding and patience throughout my graduate work.

TABLE OF CONTENTS

TABLE OF CONTENTS (continued)

LIST OF FIGURES

LIST OF FIGURES (continued)

SUMMARY

This study presents a heuristic procedure for the scheduling of activities in a cyclic project--a project which contains groups (cycles) of activities that are repeated a number of times.

When the precedence relationships of the project preclude the realization of all activities without interruptions, as they progress from one cycle to the next, it is desirable to promote the continuity of work for the activities in which the cost of interruption is more significant. These interruptions are frequently found in projects scheduled under the basic assumption of this problem, that the project must be completed by its earliest possible completion time.

This procedure produces schedules which have total interruption costs which tend to be at or near the minimum, while taking into account the varying interruption costs among activities.

CHAPTER I

INTRODUCTION

Definition of the Problem

The objective of this study is the scheduling of activities in a cyclic project, when the minimization of the cost associated with the interruption of the activities when they progress from one cycle to the next is an important factor.

A cyclic project is defined as one which contains groups (cycles) of activities that are repeated a number of times. The common cyclic project contains certain unique, or non-cyclic, starting and ending activities in addition to the cyclic portion of the project. This study deals with the cyclic portion of such projects, although the term "cyclic projects" will be used throughout. An example of a cyclic project is the construction of several identical houses. In this case, each house is a cycle of the project, and each activity necessary to the construction of one house will be repeated as many times as the number of houses in the project.

In general, each activity in the project has to be performed as many times as the number of cycles in the project and, obviously, in a particular cycle the activity cannot be started until all of its predeces-

sors in the same cycle are completed and, under the assumptions of this research, until the same activity has ended in the previous cycle. Fisher and Nemhauser (4) describe the network representation of a cyclic project as a set of networks stacked one upon the next with interconnections between the same activities in adjacent cycles.

In the scheduling of cyclic projects, it would be desirable that each activity be scheduled without interruptions; that is, the start of the activity in each cycle should be scheduled immediately following its completion in the previous cycle. This is important because every time the activity is interrupted, a cost is incurred. This cost can be due to: (1) a new setup, such as when a subcontractor moves his crew to some other project and later returns to start on the following cycle; (2) idle time in the project, if the personnel and/or equipment required to execute the activity cannot be utilized elsewhere; and (3) administrative expenses, such as the releasing of personnel after completion of a cycle and the rehiring of personnel later.

However, in most of the cases, precedence relationships of the project preclude the possibility of scheduling all activities without interruptions, and therefore, it becomes worthwhile to establish a scheduling procedure which attempts to minimize the cost due to interruptions in the project.

An interruption, as the term is used in this research, refers to a lapse of time between the completion of an activity in one cycle and

the start of the same activity in the following cycle. Interruptions within a given cycle are not considered, as no activity is scheduled to start on a given cycle until all predecessors on that cycle are complete.

Cyclic projects are frequently found in construction works, production, research experiments, and many other common situations, and are deserving, therefore, of being objects of study.

Purpose of the Research

As stated earlier, the purpose of the research was to establish a procedure for the scheduling of activities in a cyclic project which attempts to minimize the cost due to interruptions of activities in the project, by taking into account the varying cost of interruptions among the different types of activities involved in the project. This may be given as:

$$\text{Minimize} \sum_{\substack{\text{all} \\ \text{activity}_{i-j}}} c_{i-j}\, n_{i-j}$$

where n_{i-j} = number of interruptions of activity i-j,

c_{i-j} = cost of one interruption of activity i-j.

The following assumptions were made in the research:

1. The primary objective in the scheduling is to complete the

entire project by its earliest completion time.

2. No duplicate crews are available for work on the project. It is therefore not possible to be working on more than one cycle of an activity at a particular time.

3. The cost of work interruption for an activity is the same regardless of where the interruption occurs. It is therefore no more desirable to have a break of continuity when we move from the first to the second cycle than when we move from the $(n-1)^{th}$ to the n^{th} cycle.

4. The cost of work interruption for an activity is the same regardless of the length of the interruption.

Since the primary objective is to complete the project by its earliest completion time, the first step toward the solution of the problem was the development of a method for the determination of the critical path that gives the minimum duration of the project. The method was designed to be applied using the network representation of only one cycle rather than requiring the networking of all cycles because this is a very difficult task even for small projects. It should be stated that the full network representation of all cycles could also be used in the computation of the critical path of the project; however, the method developed in this study capitalizes on the structure of cyclic projects to identify the project critical path more efficiently.

Next, the research was directed toward the development of a method for the computation of earliest and latest times for the activities

in each cycle, also without requiring the graphical representation of all cycles of the project. These times are important data to the scheduling because they define the interval of time the activity can be scheduled in a certain cycle and still be within the restrictions of the critical path.

With the means established for obtaining those data, the study was concentrated on the establishment of the steps for the scheduling of activities. This scheduling is purely a combinatorial problem since in many cases when a tentative scheduling favoring a particular activity with respect to the number of interruptions is attempted, it results in a bad schedule for another activity or for a group of activities. Depending on the characteristics of the project, on the number of activities involved in it, and on the number of cycles, the problem can be computationally difficult, and no optimization technique currently available is feasible to handle it. For that reason a heuristic procedure was developed which provides a "good" solution to the problem, and which can be easily applied without requiring a large computational effort. The approach will be of great utility in the planning phase as well as in the review phase of the project.

An algorithm was also established for the computation of the apparent lower bound in the number of interruptions for an activity, which is the apparent number of times the activity would be interrupted if it were scheduled without considering the remaining activities. From

these numbers can be derived the apparent lower bound in the total cost of interruptions in the project, which can be used as a measure of effectiveness for the solution yielded by the use of the final procedure developed in this research.

The next chapter will present a survey of this procedure and a detailed discussion of its components.

Review of the Literature relating to This Subject

Only one paper was found in the literature surveyed dealing with the same type of problem considered in this research. In a Thesis by Burney (3), a technique was developed for the maximization of work continuity in the scheduling of a single activity in a cyclic project, particularly a subcontracted activity that allows one to determine the number of work interruptions for the activity without requiring the development of a detailed work schedule. He showed this technique to be of particular interest in the planning phase of the project.

A few papers like the ones by Burgess and Killebrew (2) and Fisher and Nemhauser (4) deal with the problem of scheduling in cyclic projects, but whereas the former is interested in the optimization of activity level variation in the project, the latter is interested in the construction of an algebraic network representation of a cyclic project. In addition, a book by Moder and Phillips (5) mentions briefly the problem of scheduling in cyclic projects. In all other literature surveyed, there

was no mention to cyclic projects.

On network scheduling techniques, there are several books of expository nature which provide good treatment of the subject. The books by Moder and Phillips (5), Wiest and Levy (6), and Battersby (1) are good examples of these.

CHAPTER II

PROCEDURE

In order to provide an overview, a condensed flowchart of the procedure established in this research is presented in Figure 1, showing the interrelations among its several components.

The following terminology is used in the description of the procedure:

ESS = earliest time the start of an activity can be scheduled in a cycle and still be within the critical path requirements.

LSS = latest time the start of an activity can be scheduled in a cycle and still be within the critical path requirements.

R = set of all activities not completely scheduled.

K = a subset of R formed by the activities which have all predecessors and all successors completely scheduled.

L = a subset of R formed by the activities which have all predecessors completely scheduled, but not all successors.

M = a subset of R formed by the activities which have all successors completely scheduled, but not all predecessors.

BS = schedule given by the backward algorithm.

FS = schedule given by the forward algorithm.

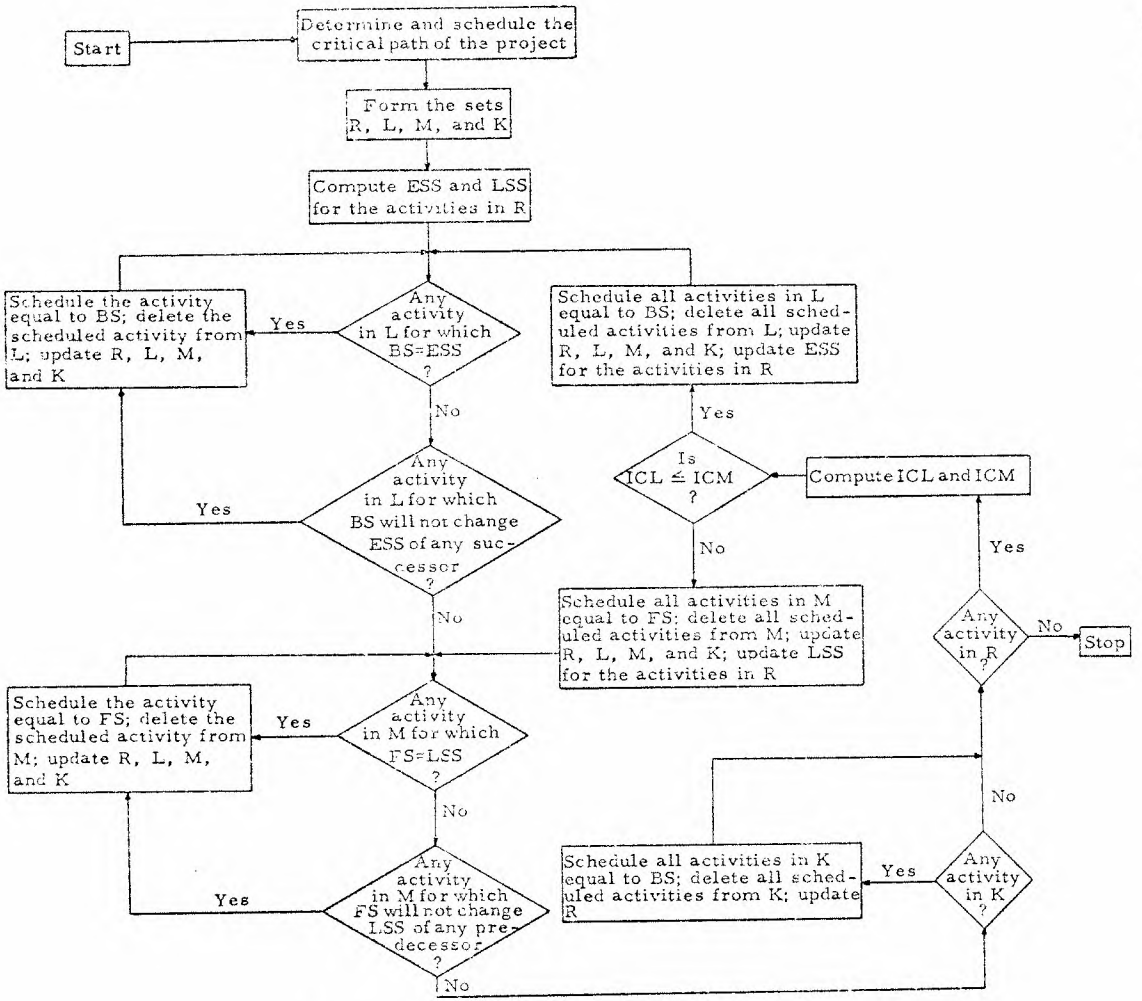


Figure 1. A Condensed Flowchart of the Overall Procedure.

Notation:

ESS = earliest time the start of an activity can be scheduled in a cycle and still be within the critical path requirements.
LSS = latest time the start of an activity can be scheduled in a cycle and still be within the critical path requirements.
R = set of all activities not completely scheduled.
K = a subset of R formed by the activities which have all predecessors and all successors completely scheduled.
L = a subset of R formed by the activities which have all predecessors completely scheduled, but not all successors.
M = a subset of R formed by the activities which have all successors completely scheduled, but not all predecessors.
BS = schedule given by the "backward algorithm."
FS = schedule given by the "forward algorithm."
ICL = increase in the apparent lower bound in the total cost of interruptions for the remaining activities to be scheduled if the activities in L are scheduled equal to BS.
ICM = increase in the apparent lower bound in the total cost of interruptions for the remaining activities to be scheduled if the activities in M are scheduled equal to FS.

ICL = increase in the apparent lower bound in the total cost of interruptions for the remaining activities to be scheduled if the activities in L are scheduled equal to BS.

ICM = increase in the apparent lower bound in the total cost of interruptions for the remaining activities to be scheduled if the activities in M are scheduled equal to FS.

The terminology used, as well as the role that each component plays in the procedure, will be explained later in this chapter, when each component of the procedure will be discussed in detail and examples will be given showing its applications.

The procedure can be described as being the assemblage of the following steps:

1. Determination of the critical path of the project by means of the "method for the determination of the critical path," and scheduling of the critical path.

2. Computation of ESS and LSS for the activities in R, in all cycles, by means of the "method for the computation of ESS and LSS."

3. Scheduling by means of the "backward algorithm" of every activity in L for which BS is equal to ESS in all cycles, or when BS is not equal to ESS, if that schedule will not change the ESS of any successor of the activity.

4. Scheduling by means of the "forward algorithm" of every activity in M for which FS is equal to LSS in all cycles, or when FS is not

equal to LSS, if that schedule will not change the LSS of any predecessor of the activity.

5. Scheduling by means of the "backward algorithm" of all activities in K.

6. For the activities in L and M for which the scheduling steps above cannot be applied, computation of ICL and ICM as explained in "Computation of ICL and ICM," and scheduling of all activities in L by means of the "backward algorithm" if ICM is greater than or equal to ICL, or otherwise, scheduling of all activities in M by means of the "forward algorithm."

Before going into a description of each component of the procedure, it should be mentioned that throughout this paper, every time the schedule of a cyclic project is to be represented graphically, a bar chart is used. The numbers on the bars, corresponding to each activity, represent the cycle of the project that is to be performed at a particular time.

Since the aim of the procedure is the minimization of the cost due to interruptions of activities in a cyclic project, the interruption of an activity in the schedule of the project is most easily shown by the use of the bar chart.

Consider as an example the cyclic project of producing six identical items. The activity sequence necessary to the production of one item (one cycle of the project) is shown in Figure 2, in a time-scaled

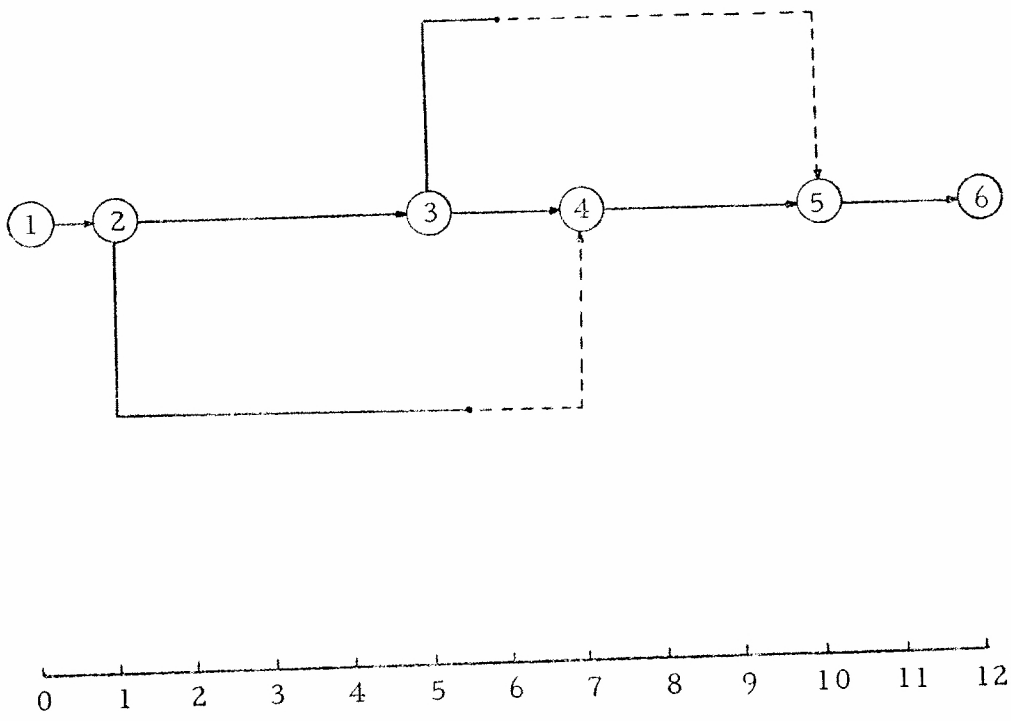


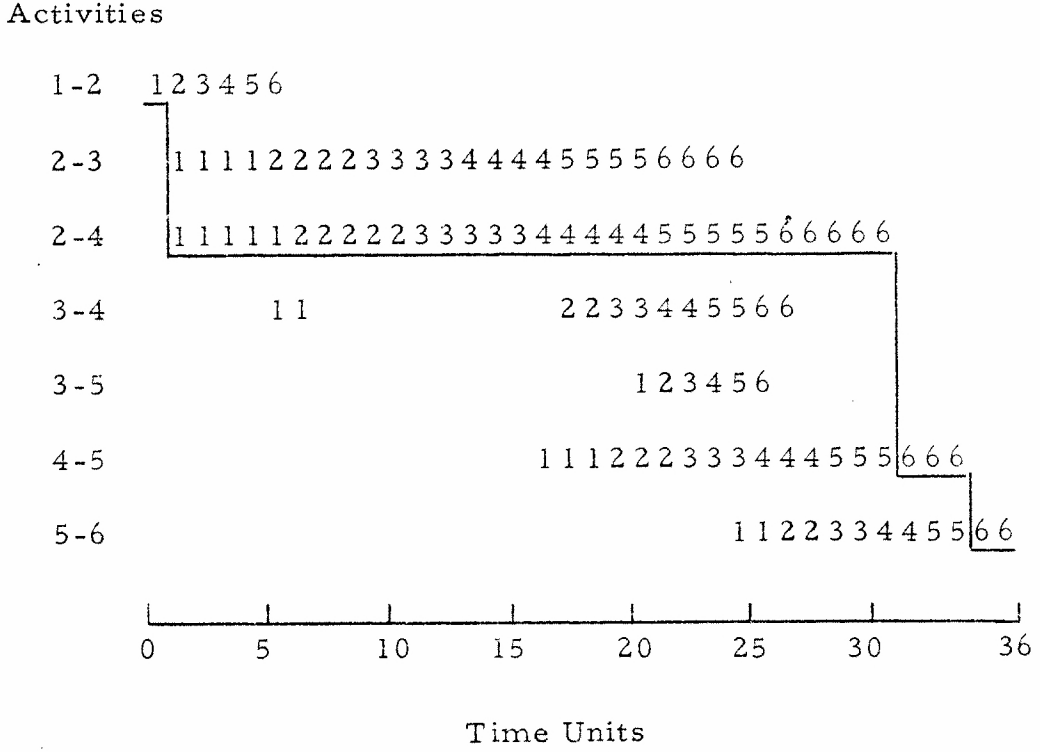
Figure 2. Time-Scaled Network of One Cycle of the Project.

network, and one possible schedule for the project is shown in a bar chart in Figure 3. In this example, Activity 3-4 is scheduled with an interruption between the first and second cycle.

Method for the Determination of the Critical Path

In a nonrepetitive project, the minimum amount of time required for its completion is the duration of the critical path. However, in a cyclic project it is not necessary that one cycle be completed before the next is begun, and therefore, the minimum amount of time required for the completion of a project with n cycles is considerably shorter than n times the duration of the critical path of one cycle.

Consider the determination of the critical path of the project of the previous example. As can be observed in Figure 3, the first cycle of Activity 2-4 could not be scheduled earlier because it could be started only after the completion of the first cycle of Activity 1-2, which was scheduled in its earliest start. Also, all other cycles of Activity 2-4 were scheduled in their earliest starts, that is, immediately following the completion of the previous cycle. Moreover, after the completion of the last cycle of Activity 2-4, to reach the end of the project in a minimum amount of time, the activities succeeding that activity had their last cycles scheduled in their earliest starts. Therefore, since all activities in the project were scheduled within the interval of time used for the scheduling of the path formed by the first cycle of Activity



Critical path = _____

Figure 3. A Schedule for the Cyclic Project Showing the Critical Path.

1-2, by all cycles of Activity 2-4, and by the last cycle of Activities 4-5 and 5-6, this path is actually the critical path of the project.

In general, the critical path of a cyclic project is formed by the first cycle of some activities, by all cycles of a certain activity, which will be called in this study "the activity dictating the minimum duration of the project," and by the last cycle of some activities in the project.

The reason for calling the activity which has all of its cycles in the critical path of the project "the activity dictating the minimum duration of the project" is that the determination of the earliest completion time of the project depends on the identification of that activity. The activities with the first cycle and with the last cycle in the critical path of the project are identified after the determination of the activity dictating the minimum duration of the project, since they are the activities in the longest path of one cycle of the project which includes that activity.

In the preceding example the activity dictating the minimum duration of the project is Activity 2-4, and the activities in the longest path of one cycle of the project which includes that activity are Activities 1-2, 4-5, and 5-6.

The only activity that can be "the activity dictating the minimum duration of the project" is the longest activity in the critical path of one cycle, or the longest activity in the project. Clearly, in some cases these two activities are the same. Generally these activities would be

a single activity, but in some cases two or more activities might simultaneously dictate the minimum duration of the project.

It should be mentioned that the longest activity of the project has a certain slack, and cycle after cycle during the scheduling of the project that slack decreases until the point that the activity becomes critical. This can be better explained by referring to the previous example. In this case the longest activity in the project is Activity 2-4, and the longest activity in the critical path of one cycle is Activity 2-3. Figure 4 shows the critical path of the project if the project were a one-cycle project, a two-cycle project, and a three-cycle project.

As can be observed in Figure 4, if the project were a one-cycle project, the activity dictating the minimum duration of the project would be Activity 2-3, and Activity 2-4 would have a slack of one time unit.

If the project were a two-cycle project, it would have two critical paths, that is, the path formed by the first cycle of 1-2, all cycles of 2-3, and the last cycle of 3-4, 4-5, and 5-6, and the path formed by the first cycle of 1-2, all cycles of 2-4, and the last cycle of 4-5 and 5-6. This time both Activities 2-3 and 2-4 would be dictating the minimum duration of the project.

If the project were a three-cycle project, the activity dictating the minimum duration of the project would be Activity 2-4, and Activity 2-3 would have a slack of one time unit. Obviously, as the number of cycles of the project are increased, the slack of Activity 2-3 increases

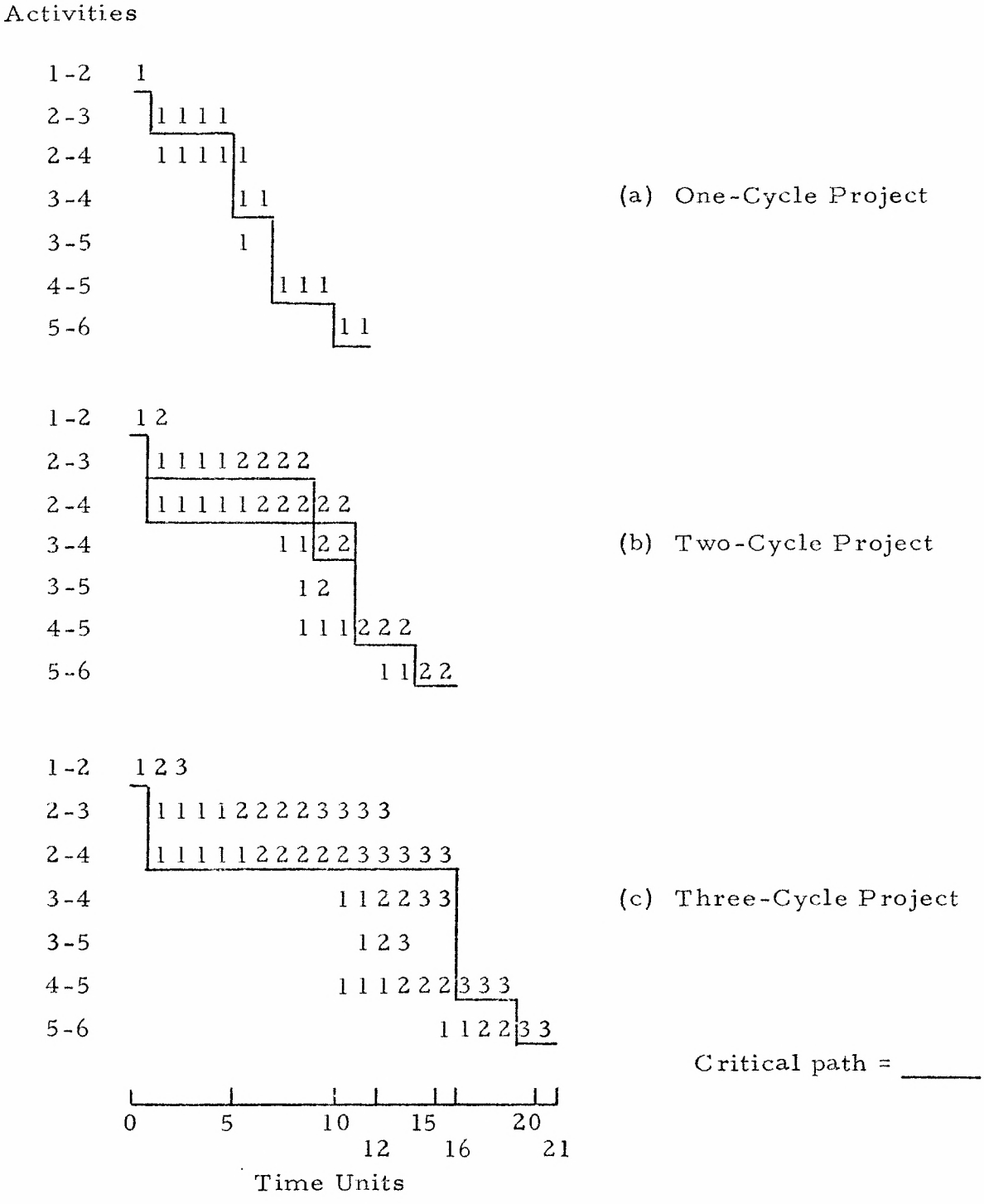


Figure 4. A Schedule Showing the Critical Path for One, Two, and Three Cycles of the Project.

more and more.

Hence, the number of cycles in the project is an important factor in the determination of the point at which the longest activity in the project starts to be the activity dictating the minimum duration of the project in place of the longest activity in the critical path of one cycle.

An index which determines the number of cycles that will take the longest activity in the project to start dictating the minimum duration of the project is defined below.

$$X = \frac{TS}{DL-DC} + 1$$

where X = index,

TS = total slack of the longest activity in the project, one cycle being considered,

DL = duration of the longest activity in the project, one cycle being considered,

DC = duration of the longest activity in the critical path, one cycle being considered.

The minimum duration of the project is dictated by the longest activity in the project if X is smaller than the number of cycles, by the longest activity in the critical path of one cycle if X is larger than the

number of cycles, and by both activities if X is equal to the number of cycles.

The ideas stated above led to the establishment of a method for the determination of the critical path of a cyclic project. The rules of this methodology are described below.

Rule 1. If the longest activity in the project is in the critical path of one cycle, the critical path of the cyclic project is formed by the first cycle of the activities preceding that activity along the critical path of one cycle, by all cycles of that activity, and by the last cycle of the activities succeeding that activity along the critical path of one cycle.

Rule 2. If the longest activity in the project is not in the critical path of one cycle, compute the index X as defined above.

(a) If X is greater than the number of cycles in the project, the critical path of the cyclic project is formed by the first cycle of the activities preceding the longest activity in the critical path of one cycle along this path, by all cycles of the longest activity in the critical path of one cycle, and by the last cycle of the activities succeeding the longest activity in the critical path of one cycle along this path.

(b) If X is smaller than the number of cycles in the project,

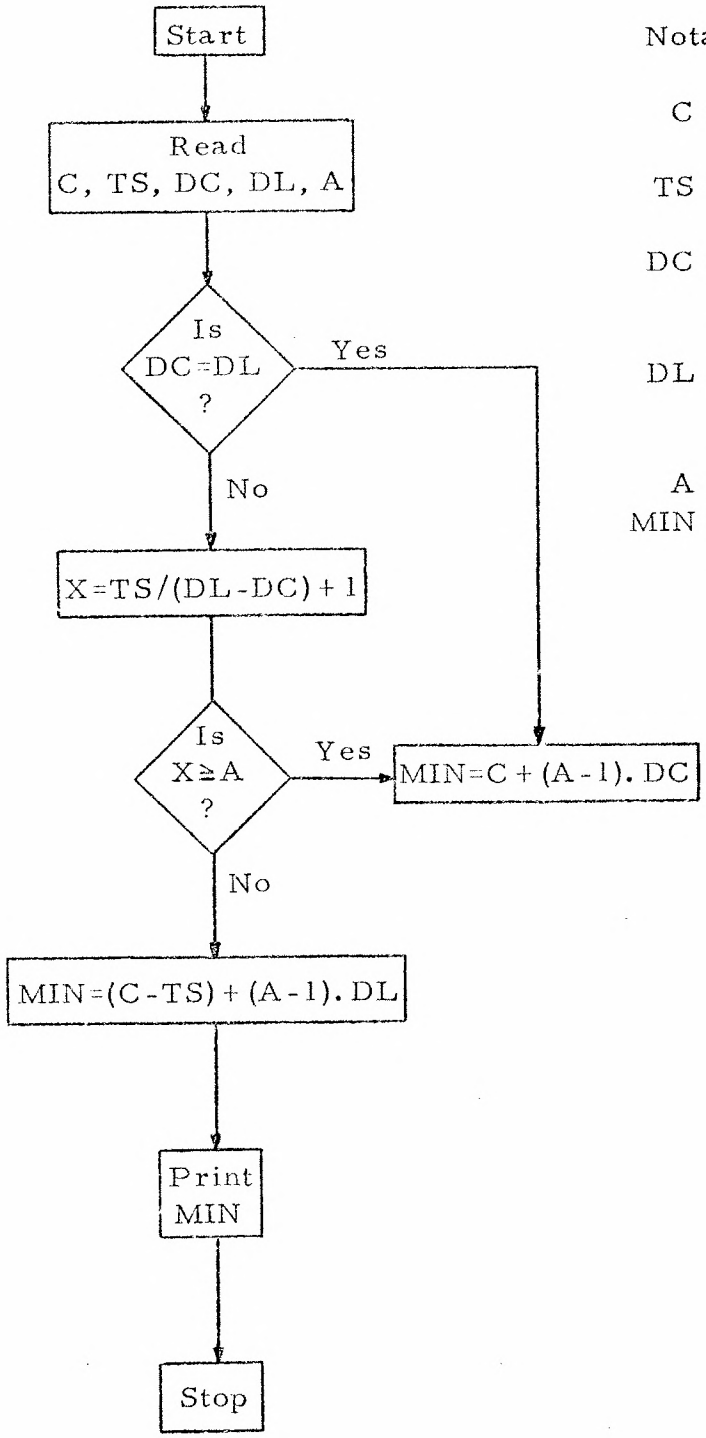
the critical path of the cyclic project is formed by the first cycle of the activities preceding the longest activity in the project along the longest path of one cycle which includes that activity, by all cycles of the longest activity in the project, and by the last cycle of the activities succeeding the longest activity in the project along the longest path of one cycle which includes that activity.

(c) If X is equal to the number of cycles in the project, the critical path of the cyclic project is multiple critical paths formed by both (a) and (b).

Figure 5 presents an algorithm for the computation of the earliest completion time of a cyclic project. This algorithm makes use of Rules 1 and 2 for determining the project critical path. It should be mentioned that, as in a nonrepetitive project, for the completion of a cyclic project by its earliest completion time, the critical path of the project must be scheduled necessarily in its earliest start. Hence, the first scheduling step of the overall procedure is to schedule the critical path activities of the project.

As an example of the use of the method, consider the determination of the critical path of three cycles of the project whose one-cycle representation was presented in Figure 2.

In this case the longest activity in the project is not in the criti-



Notation:

C = length of the critical path of one cycle.
TS = total slack of the longest activity (one cycle).
DC = duration of the longest activity in the critical path (one cycle).
DL = duration of the longest activity in the project (one cycle).
A = number of cycles.
MIN = minimum duration of A cycles.

Figure 5. Algorithm for the Computation of the Earliest Completion Time of A Cycles in a Cyclic Project

cal path of one cycle, and the index X must be computed. This computation yields a result of 2.0. Since the index X is smaller than the number of cycles in the project, by Rule 2(b), the critical path of the cyclic project is formed by the first cycle of Activity 1-2, by all cycles of Activity 2-4, and by the third cycle of Activities 4-5 and 5-6. This is represented in Figure 4(c).

If the problem were the determination of the critical path of two cycles of the same project, the index X would be equal to the number of cycles in the project, and Rule 2(c) would be used to obtain what is represented in Figure 4(b).

As an example of the use of the algorithm in Figure 5, consider the computation of the earliest completion time of seven cycles of the project. The logic of the algorithm yields

$$MIN = (C-TS) + (A-1)DL$$

$$= (12-1) + (6-1)5 \qquad \text{(Ref. page 21 for notation)}$$

$$= 36 \text{ time units.}$$

Method for the Computation of ESS and LSS

This method is used in the computation of the earliest and latest times that the start of an activity can be scheduled in each specific cycle of the project, as differentiated from ES and LS for the single cycle project. It involves a forward pass for the computation of the earliest

times and a backward pass for the computation of the latest times. For its application it is required that for each activity i-j, that is, an activity with predecessor event i and successor event j, j be always greater than i.

In the description of the method, the term EF is used to represent the earliest finish time for an activity in a cycle. The forward pass rules for the computation of the ESS are summarized below.

1. Start computing the ESS for the activity with the smallest i plus j. After that, for the activity with the second smallest i plus j, and so forth, until the activity with the largest i plus j is reached. In case of a tie, pick the activity with the smaller i.

2. Work on each activity from the first cycle to the last cycle.

3. The ESS for the first cycle of the activity is the latest EF of all of its predecessors, in that cycle, plus one.

4. From the second cycle to the last cycle, the ESS of the activity, in each cycle, is either the latest EF of all of its predecessors, in the cycle in question, plus one, or the ESS of the activity, in the previous cycle, plus its duration. The ESS is the largest of those two numbers.

5. To compute the EF of an activity, at each cycle, add the duration of the activity minus one to the ESS of the activity in the cycle in question.

The backward pass rules for the computation of the LSS are

summarized below.

1. Start computing the LSS for the activity with the largest i plus j. After that, for the activity with the second largest i plus j, and so forth, until the activity with the smallest i plus j is reached. In case of a tie, pick the activity with the larger j.

2. Work on each activity from the last cycle to the first cycle.

3. The LSS for the last cycle of the activity is the earliest LSS of all of its successors, in that cycle, less the duration of the activity.

4. From the next to the last cycle to the first cycle, the LSS of the activity in each cycle is either the earliest LSS of all of its successors in the cycle in question less the duration of the activity, or the LSS of the activity in the following cycle less its duration. The LSS is the smallest of those two numbers.

Consider again the project whose one-cycle representation was presented in Figure 1. As an example the method was used in the computation of the ESS and LSS for the activities in this project, and the results are given in Figure 6. Notice that the critical path of the project has the ESS equal to the LSS as should be expected.

Forward and Backward Algorithms

Consider a scheduling procedure that starts scheduling the first cycle of the activity as late as possible, that is, in its LSS. If there is a possibility of progressing to the second cycle without interrupting the

Activity	Cycle											
	1		2		3		4		5		6	
	ESS	LSS	ESS	LSS	ESS	LSS	ESS	LSS	ESS	LSS	ESS	LSS
1-2	1	1	2	6	3	11	4	16	5	21	6	25
2-3	2	6	6	10	10	14	14	18	18	22	22	26
2-4	2	2	7	7	12	12	17	17	22	22	27	27
3-4	6	15	10	18	14	21	18	24	22	27	26	30
3-5	6	24	10	26	14	28	18	30	22	32	26	34
4-5	8	17	12	20	17	23	22	26	27	29	32	32
5-6	11	25	15	27	20	29	25	31	30	33	35	35

Figure 6. ESS and LSS for the Project Whose One-Cycle Representation Was Presented in Figure 2.

activity, or in other words, if the ESS of the second cycle less the LSS of the first cycle is less than or equal to the duration of the activity, the procedure schedules the second cycle of the activity in a time equal to the LSS of the first cycle plus the duration of the activity; otherwise, it schedules the second cycle of the activity as late as possible, or in its LSS. When it is possible to progress from the first to the second cycle continuously, the procedure verifies the possibility of progressing from the second to the third cycle without interrupting the activity, that is, if the ESS of the third cycle less the time that the activity was scheduled to start in the second cycle is less than or equal to the duration of the activity, and otherwise, it schedules the third cycle of the activity in its LSS. The same idea is used for all other cycles of the project; that is, when it is possible to progress to a cycle without interrupting the activity, the procedure will schedule the activity in that way; otherwise, it will schedule the activity as late as possible to try to progress to the following cycle without interruption. This scheduling procedure is referred to as the "forward algorithm."

Similarly, consider a scheduling procedure that starts scheduling the last cycle of an activity in its ESS and attempts to progress from cycle to cycle without interrupting the activity and, when it is not possible to do so, schedules the activity in its ESS. This scheduling procedure is referred to as the "backward algorithm." These two algorithms are mathematically expressed as follows:

Forward algorithm:

$$S1 = LSS1$$

$$SN = S(N-1) + D \quad \text{, if } ESSN - S(N-1) \leq D$$
$$= LSSN \quad \text{, otherwise}$$

$$N = 2, 3, \ldots, A$$

Backward algorithm:

$$SA = ESSA$$

$$SN = S(N+1) - D \quad \text{, if } S(N+1) - LSSN \leq D$$
$$= ESSN \quad \text{, otherwise}$$

$$N = (A-1), (A-2), \ldots, 1$$

where A = number of cycles,

D = duration of the activity,

ESSN = ESS in the N^{th} cycle,

LSSN = LSS in the N^{th} cycle,

SN = scheduled start in the N^{th} cycle.

Figures 7 and 8 present the forward and backward algorithms in flowchart form.

The logic underlying the forward algorithm is as follows:

Starting the first of several cycles at its LSS can only reduce the possible number of interruptions, since the only potential schedule

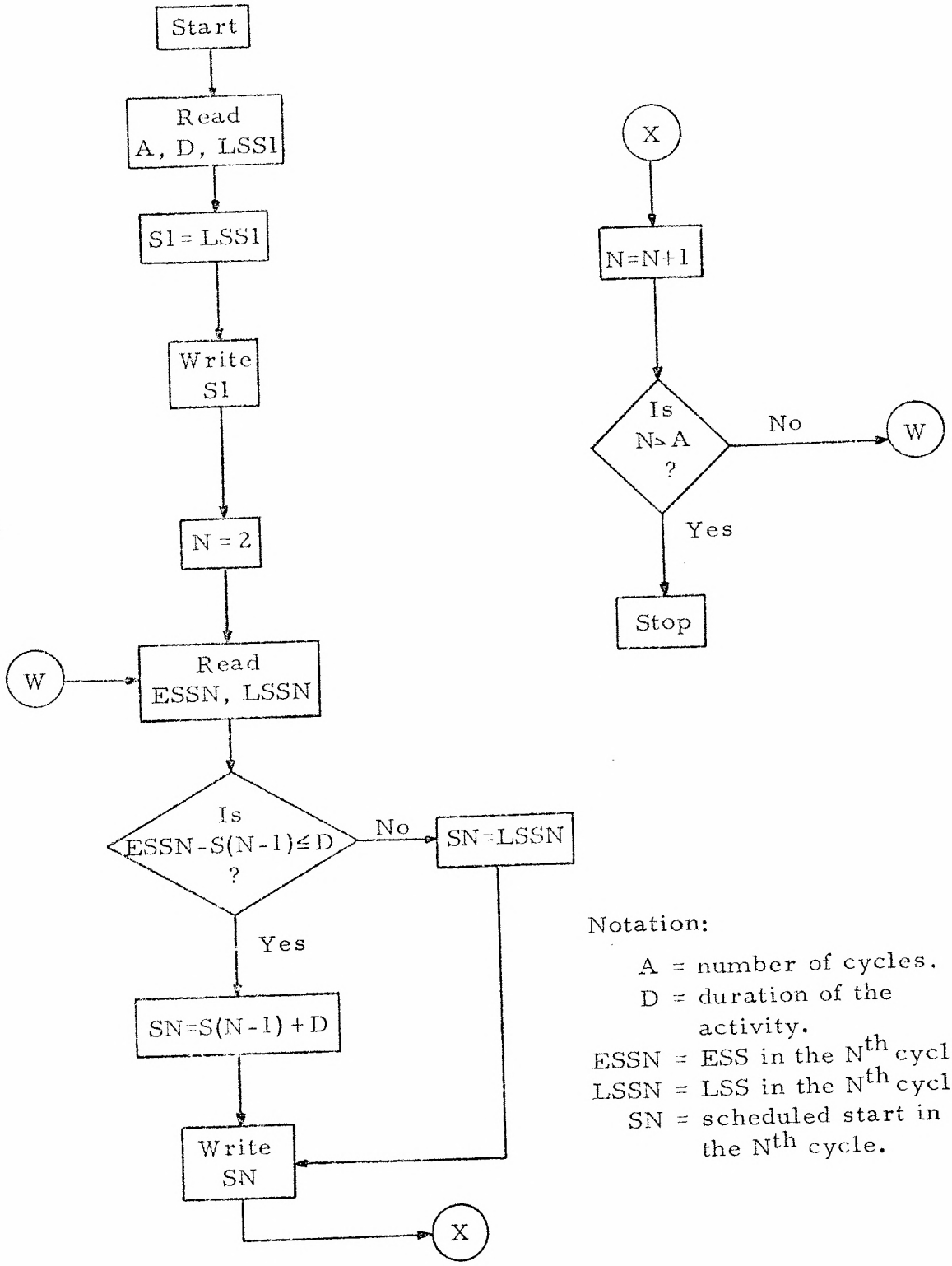


Figure 7. Forward Algorithm.

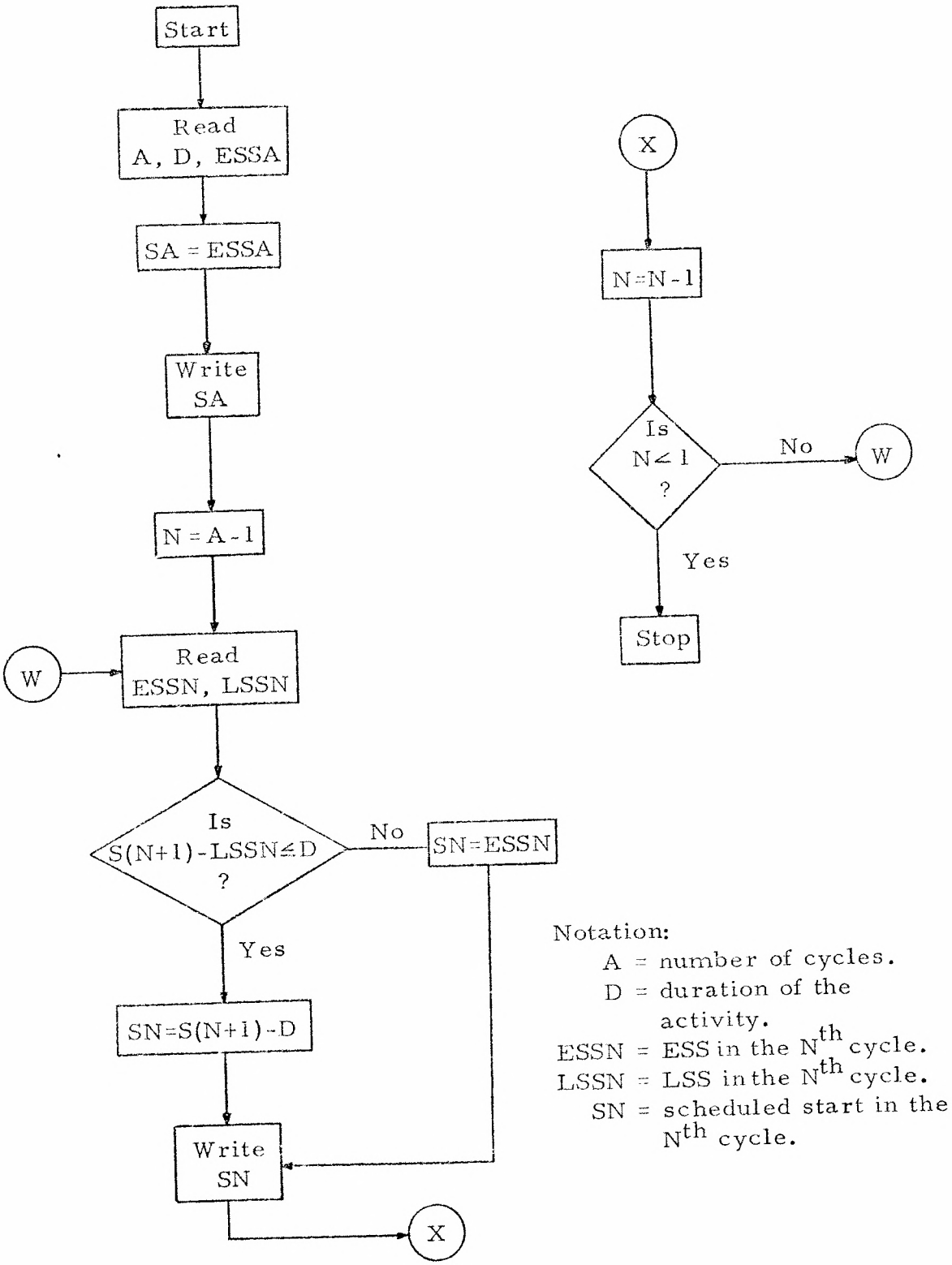


Figure 8. Backward Algorithm.

positions which are eliminated thereby are those involving cycles which are already being done continuously from Cycle 1.

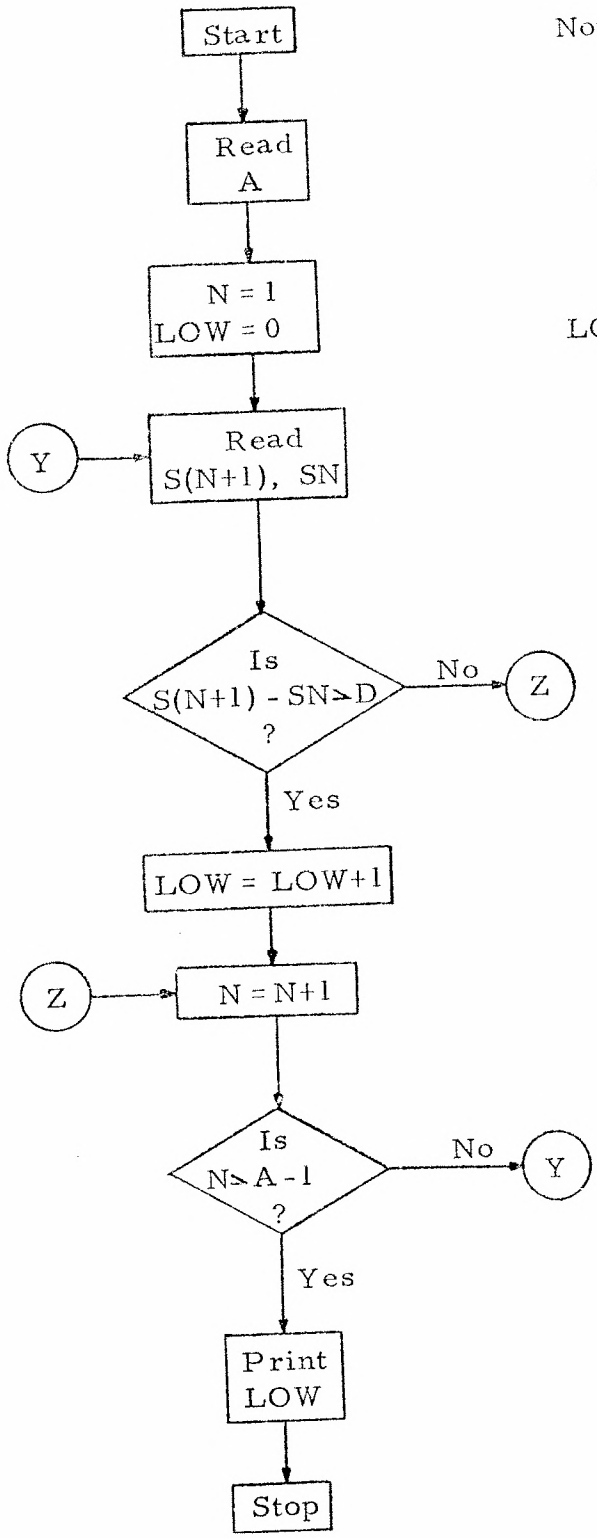
Scheduling Cycle N earlier than its LSS can only cause one interruption in the following cycles, in that the following cycle, (N+1), can still be scheduled at its LSS if desired, creating a possible interruption only between Cycle N and Cycle (N+1). If scheduling a cycle earlier than its LSS is necessary to avoid an interruption between Cycle N and Cycle (N-1), then the action of so scheduling trades off the elimination of one otherwise certain interruption, between Cycles N and (N-1), at the cost of possibly creating one interruption between Cycles N and (N+1).

This logic can be easily placed in its reverse form to explain the design of the backward algorithm.

The apparent lower bound in the number of interruptions for an activity can be derived from either of those algorithms and is equal to the number of times (SN+1 - SN) is greater than D, for N = 1, 2, . . . , (A-1). Figure 9 presents an algorithm for the computation of that number.

As an example of the use of these three algorithms, consider the problem of determining the apparent lower bound in the number of interruptions for Activity 3-4 in the project whose one-cycle representation was presented in Figure 2.

The schedule given by the forward algorithm is S1 = 15, S2 = 18,



Notation:

A = number of cycles.
D = duration of the activity.
SN = scheduled start in the N^{th} cycle, given by either the forward algorithm or the backward algorithm.
LOW = apparent lower bound in the number of interruptions.

Figure 9. Algorithm for the Computation of the Apparent Lower Bound in the Number of Interruptions for an Activity.

S3 = 20, S4 = 22, S5 = 24, S6 = 26, which shows an interruption from the first to the second cycle, since (S2-S1) is greater than the duration of the activity. Hence, the apparent lower bound in the number of interruptions for that activity is one.

The schedule given by the backward algorithm is S6 = 26, S5 = 24, S4 = 22, S3 = 20, S2 = 18, S1 = 6, which also shows that the apparent lower bound in the number of interruptions for the activity in question is one, as should be expected.

Scheduling Steps of the Overall Procedure

Suppose that, during the scheduling of the project, all predecessors of an activity happen to be completely scheduled; that is, the activity is in the subset L. If that activity is scheduled by means of the backward algorithm, it is being interrupted a number of times equal to the apparent lower bound in the number of interruptions for the activity. Also, if that activity is scheduled in its ESS, all cycles, it is being placed as far to the left of the bar chart as possible; and therefore, this scheduling leaves the most possible slack available for the scheduling of the remaining activities.

Hence, the second scheduling step is to schedule by means of the backward algorithm every activity in the subset L for which the schedule given by the backward algorithm is equal to the ESS of the activity in all cycles.

Similarly, if an activity with all successors completely scheduled, that is, an activity in the subset M, is scheduled by means of the forward algorithm, it is being interrupted a number of times equal to the apparent lower bound in the number of interruptions. Also, if that activity is scheduled in its LSS, all cycles, it is being placed as far to the right of the bar chart as possible, and therefore, this scheduling leaves the most possible slack available for the scheduling of the remaining activities.

Hence, the third scheduling step is to schedule by means of the forward algorithm every activity in the subset M for which the schedule given by the forward algorithm is equal to the LSS of the activity in all cycles.

A consequence of these two steps is that both the starting and ending activities, that is, the activities starting and ending a cycle, will be scheduled by the backward and forward algorithms, respectively.

The starting activities can be considered activities with all predecessors completely scheduled. Since a starting activity has no predecessors, the ESS of the first cycle is equal to zero and, by the forward pass rule for the computation of the ESS number four, the ESS for each of the other cycles is equal to the ESS of the previous cycle plus the duration of the activity. It can be easily verified that the schedule given by the backward algorithm is equal to the ESS of the activity in all cycles.

The ending activities can be considered activities with all successors completely scheduled, and for these activities the schedule given by the forward algorithm is always equal to the LSS in all cycles, as can also be verified.

Before going into the description of the remaining steps, it should be observed that when an activity is scheduled, it can cause only a change in the ESS of its successors and a change in the LSS of its predecessors, independent of the procedure used in its scheduling. This can be understood by recalling the forward pass and backward pass rules for the computation of the ESS and LSS.

Similarly, if an activity when scheduled is not causing any change in the ESS of its successors, that schedule is not changing the ESS of any other activity in the project. Likewise, if an activity when scheduled is not causing any change in the LSS of its predecessors, that schedule is not changing the LSS of any other activity in the project.

If an activity in the subset L is scheduled in its ESS in all cycles, this schedule obviously will not cause any change in the ESS of its successors. Also, no change can occur in the LSS of its predecessors since they are already scheduled.

However, if an activity in the subset L is scheduled in such a way that the resulting schedule does not coincide with the ESS of the activity in all cycles, this may cause an alteration in the ESS of the successors of the activity and, hence, in the ESS of the successors of

the successors, and so forth.

In some cases when more than one activity has the same successors, that is, when more than one activity is merging to the same event in a one-cycle representation of the project, one of these activities may be scheduled in such a way that the resulting schedule does not coincide with its ESS in all cycles and does not cause any change in the ESS of the successors. This is due to a slack in the path formed by all cycles of the activity.

Hence, in some cases an activity in the subset L may be scheduled in such a way that the resulting schedule does not coincide with the ESS of the activity in all cycles and does not increase the apparent lower bound in the number of interruptions for the remaining activities to be scheduled. Recall that this number is a function of the ESS and LSS of the activities.

This suggests that one should verify for every activity in the subset L, in the case that the schedule given by the backward algorithm does not coincide with the ESS of the activity in all cycles, if that schedule will not cause a change in the ESS of any successor of the activity. If no change will occur, the activity is scheduled by means of the backward algorithm. This is the fourth step established.

The same kind of reasoning used above would show that, in some cases, an activity in the subset M may be scheduled in such a way that the resulting schedule does not coincide with the LSS of the activity

in all cycles and does not increase the apparent lower bound in the number of interruptions for the remaining activities to be scheduled.

Hence, the fifth step established is to schedule by means of the forward algorithm every activity in the subset M, for which, although the schedule given by the algorithm does not coincide with the LSS of the activity in all cycles, that schedule will not change the LSS of any predecessor of the activity.

Consider now an activity which has all predecessors and all successors completely scheduled, that is, an activity in the subset K. Obviously, the activity could be scheduled by any means without causing a change in the ESS and LSS of the activities in the project. However, it would be wise to use either the forward or the backward algorithms, since they attempt to schedule an activity without interrupting it. Therefore, the sixth step established is to schedule by means of the backward algorithm every activity in the subset K, thereby conserving all possible slack.

It may happen that at a certain stage of the scheduling, none of the steps already described can be applied. The step that was established to handle this situation is to schedule either all activities in the subset L by means of the backward algorithm or all activities in the subset M by means of the forward algorithm, depending on which subset will give, at that stage, the lowest increase in the apparent lower bound in the total cost of interruptions for the remaining activities to

be scheduled. This is the seventh step.

Computation of ICL and ICM

As just stated, the seventh step is based on the decision between scheduling all of the activities in the subset L by means of the backward algorithm or scheduling all of the activities in the subset M by means of the forward algorithm. This decision is made by comparing the ICL with the ICM.

In general, the increase in the apparent lower bound in the total cost of interruptions for the remaining activities to be scheduled, if all of the activities in one of the subsets L or M are simultaneously scheduled, is given by:

$$\sum_{\substack{\text{all activities } i\text{-}j \\ \in R}} (a_{i-j} - b_{i-j})\, c_{i-j}$$

where c_{i-j} = cost of one interruption of activity i-j,

b_{i-j} = apparent lower bound in the number of interruptions for the activity i-j before the scheduling of the subset,

a_{i-j} = apparent lower bound in the number of interruptions for the activity i-j after the scheduling of the subset.

Recall that the apparent lower bound in the number of interrup-

tions for an activity is obtained from the schedules given by the forward and backward algorithm.

It should be mentioned that the increase in the apparent lower bound in the number of interruptions for the remaining activities to be scheduled is due to the fact that, when the seventh step is applied, either all activities in the subset L are not being scheduled in their ESS in all cycles, or all activities in the subset M are not being scheduled in their LSS in all cycles.

Hence, when this step is applied, the scheduling of the subset L implies a revision in the ESS of the activities in R, and the scheduling of the subset M, a revision in the LSS.

Logic of the Overall Procedure

At each stage of the scheduling, the procedure tries to schedule the activities in R by applying one of the first six steps. When it is not possible to do so, the procedure uses the seventh step, and after identifying the revised contents of sets R, L, M, and K, it returns to its attempt to use steps two through six.

Application and Results

In order to make some judgment about its effectiveness, the overall procedure was used in the scheduling of ten example projects. The results are given in Figure 10.

Figure 10 includes, in addition to the apparent lower bound in

Project Number	Number of Activities	Number of Cycles	Ap. Lower Bound in the Total No. of Interruptions	Total No. of Inter Yielded by the Procedure	Known Lower Bound in the Total Cost of Interruptions	Ap. Lower Bound in the Total Cost of Inter. (Min)	Total Cost of Inter. Yielded by the Proc. (Proc)	Ap. Upper Bound in the Total Cost of Inter. (Max)	Known Upper Bound in the Total Cost of Interruptions	$\frac{\text{Proc - Min}}{\text{Max - Min}}$
1	7	7	4	7	6	10	14	84	84	0.054
2	12	8	5	5	2	8	8	175	175	0.000
3	28	5	11	21	35	40	58	264	264	0.080
4	7	6	1	1	0	2	2	63	65	0.000
5	6	7	0	0	0	0	0	74	74	0.000
6	7	10	5	10	9	15	25	134	134	0.074
7	7	5	2	3	2	4	5	46	46	0.022
8	8	8	12	14	11	20	22	136	136	0.015
9	7	7	7	10	6	14	18	79	79	0.050
10	14	6	6	10	4	12	20	144	144	0.060

Figure 10. Evaluation of the Effectiveness of the Overall Procedure

the total cost of interruptions in the project, which is a by-product of the procedure, the apparent upper bound in the total cost of interruptions in the project. This apparent upper bound in the total cost of interruptions is calculated as described in Appendix A. The upper bound represents a schedule tendency which frequently occurs in cyclic projects. This is true because the upper bound reflects an attempt to schedule each cycle with activities in the same relative position to each other as in every other cycle. This tendency occurs in practice because of the administrative ease of issuing and controlling such a schedule. Thus, the upper bound is one realistic measure to use in evaluating the effectiveness of the procedure.

These apparent lower and upper bounds in the total cost give a range in which the actual total cost apparently must fall, and does provide a basis for evaluating the effectiveness of the overall procedure. Since this evaluation involves the cost as well as the number of interruptions, the apparent lower and upper bounds in the total number of interruptions, and the total number of interruptions in the solution yielded by the use of the procedure, are also given in Figure 10. This helps to demonstrate that the apparent effectiveness of the overall procedure is not attributable to a particular assignment of costs in the example projects.

In examining the data in Figure 10, the total cost of interruptions in the solution yielded by the use of the procedure can be com-

pared with the apparent lower bound in the total cost of interruptions in the project. If the former is not excessively higher than the latter, the solution can be considered a good one. However, although the closeness of those costs is an evidence of a good solution, a large difference is not a proof of a bad solution, since in most of the cases, the apparent lower bound in the total cost of interruptions is not an attainable cost in that it is obtained by assuming each activity to be independent of all others. Also, it should be mentioned that, as well as the apparent lower bound, the apparent upper bound in the total cost of interruptions in the project is not an attainable cost in all cases.

It will be noted in every example that the results of the procedure are extremely close to the apparent lower bound and extremely distant from the apparent upper bound.

Because no proof is offered to verify that the apparent lower bound is an absolute minimum, or that the apparent upper bound is an absolute maximum, known lower and upper bounds are also given in Figure 10. The known lower bound is calculated by determining the number of times a given activity has its ESS greater than the LSS plus the activity duration in the previous cycle each such instance will require an interruption of that activity. These interruptions, multiplied by the cost of one interruption of that activity and, summed over all activities, give a known lower bound on the cost of interruption. The known upper bound is calculated by multiplying the number of cycles

minus one (the maximum interruptions of a given activity) by the cost of one interruption for the activity and summing this over all except the activity dictating the minimum duration of the project. In the example project, the evaluation of the procedure appears to be essentially unaffected by the small difference in the apparent and known bounds.

Appendix B shows the procedure being applied to three different projects. The first application of the procedure is described in detail. In the other two, a summary of the application is given.

To make possible the use of the procedure in the scheduling of larger projects, Appendix C presents the overall procedure in a detailed flowchart, ready to be programmed for computer use. It should be pointed out that the application of the procedure to larger problems will more precisely evaluate its effectiveness.

CHAPTER III

CONCLUSIONS AND RECOMMENDATIONS

The procedure established in this research is capable of providing a good solution to the problem of scheduling in cyclic projects, when the minimization of the total cost of interruptions in the project is an important factor, although no claim of optimality is made.

The several methods and algorithms, developed in this study to be used as subroutines of the overall procedure, certainly will be valuable for research in this problem, or related problem areas. As an example, the "Algorithm for the computation of the earliest completion time of a cyclic project" could be used in evaluating the possibility of executing a certain cyclic project within a previously established length of time. Another example could be the use of the "Method for the computation of ESS and LSS" by someone interested in researching the problem of resource allocation in cyclic projects. Many other examples could be given, showing the importance of the groundwork provided by this study for future work in the area.

Recommendations

The following recommendations are made regarding future improvements in the work done in this research:

1. Development of steps to apply the procedure at an update point of the project to cope with an off-schedule situation. This must be capable of being applied at any point of the project.

2. Adaptation of the procedure for its application to cyclic projects in which the identical activities may have different durations in different cycles.

3. Establishment of means for the evaluation of the actual cost yielded by the use of the procedure, since there is a cost associated with the application of the procedure itself.

4. Establishment of steps to examine potential improvements in the final schedule yielded by the use of the procedure, within the bounds of number of interruptions in the schedule, with respect to other measures of effectiveness, such as allocation of resources.

5. Computerization of the procedure, to aid in both application to actual projects and in solution of larger examples in sufficient number to improve the precision with which the effectiveness of the procedure can be evaluated.

APPENDIX A

ALGORITHM FOR THE COMPUTATION OF THE APPARENT UPPER BOUND IN THE NUMBER OF INTERRUPTIONS FOR AN ACTIVITY

APPENDIX A

ALGORITHM FOR THE COMPUTATION OF THE APPARENT UPPER BOUND IN THE NUMBER OF INTERRUPTIONS FOR AN ACTIVITY

The algorithm consists of two steps. The first is the establishment of a schedule allowing for maximum interruption of the activity. The next is the computation of the number of times that the activity is interrupted. These two steps are described below.

Notation:

A = number of cycles,

D = duration of the activity,

ESSN = ESS in the N^{th} cycle,

LSSN = LSS in the N^{th} cycle,

SN = scheduled start in the N^{th} cycle,

UPP = apparent upper bound in the number of interruptions for the activity.

Step 1:

S1 = ESS1

$$
\begin{aligned}
SN &= S(N-1) + (D+1) \quad , \text{ if } ESSN \leq S(N-1) + (D+1) \leq LSSN \\
&= S(N-1) + D \quad , \text{ if } S(N-1) + (D+1) \succ LSSN \\
&= ESSN \quad , \text{ if } S(N-1) + (D+1) \prec ESSN
\end{aligned}
$$

$N = 2, 3, \ldots, A$

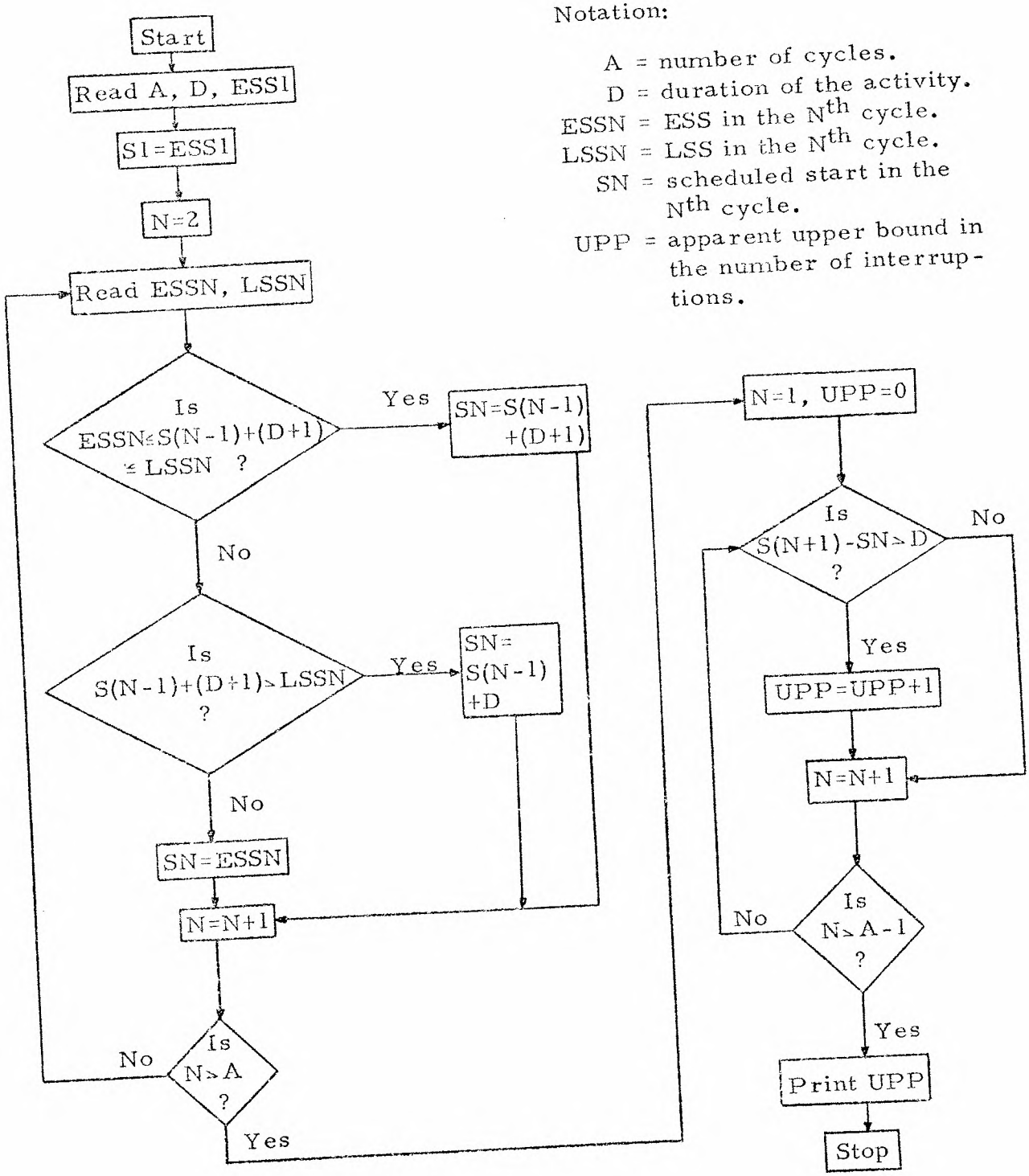
Step 2:

UPP = number of times $(S(N+1) - SN)$ is greater than D,

for $N = 1, 2, \ldots, (A-1)$.

Figure 11 presents the algorithm in flowchart form.

It should be pointed out that, despite the appearance that the apparent upper bound in the number of interruptions for an activity should be equal to the number of cycles in the project less one, in some cases, the apparent upper bound in the number of interruptions is smaller than this. That difference is due to the requirement for completion of the project in its earliest completion time. This means that when the activity dictating the minimum duration of the project is one which has total slack within a single-cycle network, this activity will not be in the same relative position in the first and last cycles, and other activities may have some contiguous cycles. Activity 2-3 in the project whose one-cycle representation is presented in Figure 2 is an example of this.



Notation:

A = number of cycles.
D = duration of the activity.
ESSN = ESS in the N^{th} cycle.
LSSN = LSS in the N^{th} cycle.
SN = scheduled start in the N^{th} cycle.
UPP = apparent upper bound in the number of interruptions.

Figure 11. Algorithm for the Computation of the Apparent Upper Bound in the Number of Interruptions for an Activity.

APPENDIX B

EXAMPLES OF THE USE OF THE PROCEDURE

APPENDIX B

EXAMPLES OF THE USE OF THE PROCEDURE

Example 1:

This is a seven-cycle project with precedence relationships as shown in Figure 12. The number on the arrows represents the cost of one interruption for the respective activity.

The application of the procedure in the scheduling of this project is shown below.

1. Determination of the critical path: Since the longest activity in the project (Activity 1-2) is in the critical path of one cycle of the project, by Rule 1 of the method, the critical path of the project is formed by all cycles of Activity 1-2 and the last cycle of the Activities 2-3, 3-5, 5-6, and 6-7.
2. Scheduling of the critical path of the project in its earliest start (first step of the procedure). See Figure 13.
3. Set R is formed by Activities 2-3, 2-4, 3-5, 4-6, 5-6, and 6-7,

   Subset L is formed by Activities 2-3 and 2-4,

   Subset M is formed by Activity 6-7,

   Subset K is empty.
4. Computation of ESS and LSS for the activities in R. The results

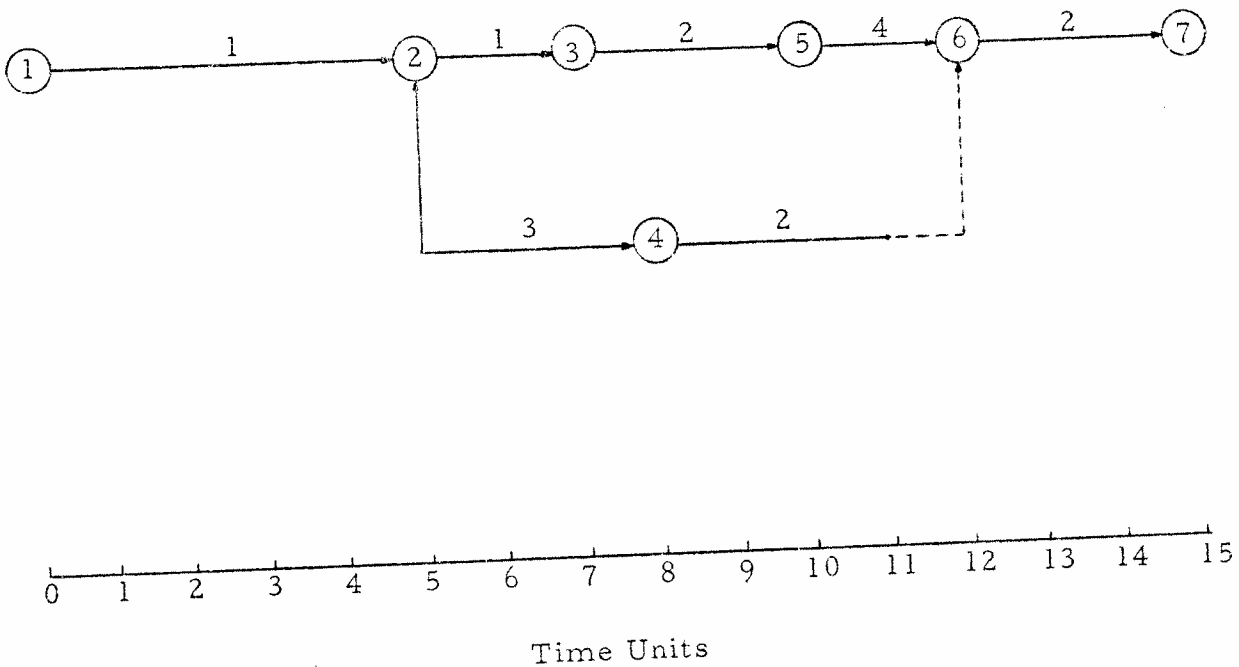


Figure 12. Time-Scaled Network of One Cycle of the Project of Example 1.

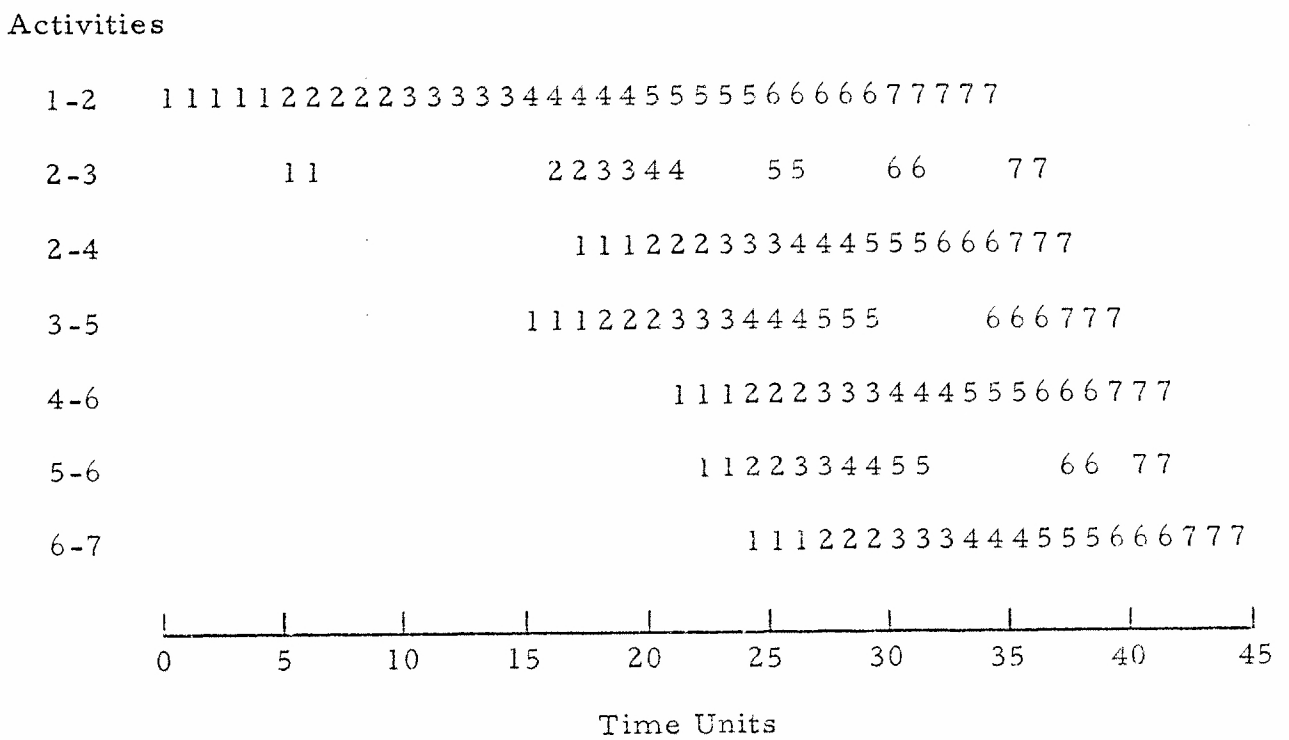


Figure 13. A Schedule for the Cyclic Project of Example 1.

are given in Figure 14.

5. Computation of the schedule given by the backward algorithm (BS) for the activities in L (Activities 2-3 and 2-4):

Activity	Cycle						
	1	2	3	4	5	6	7
2-3	12	14	16	27	29	31	36
2-4	18	21	24	27	30	33	36

Hence, no activity in L has the schedule given by the backward algorithm equal to its ESS in all cycles. Step 2 does not apply.

6. Verification for every activity in L (Activities 2-3 and 2-4) if the schedule given by the backward algorithm (BS) will not cause a change in the ESS of any successor of the activity.

Successors of Activity 2-3: Activity 3-5,

Computation of ESS for Activity 3-5 if Activity 2-3 were scheduled equal to BS:

	Cycle						
	1	2	3	4	5	6	7
ESS	14	16	18	29	31	33	38

Activity	Cycle													
	1		2		3		4		5		6		7	
	ESS	LSS	ESS	LSS	ESS	LSS	ESS	LSS	ESS	LSS	ESS	LSS	ESS	LSS
2-3	6	18	11	21	16	24	21	27	26	30	31	33	36	36
2-4	6	19	11	22	16	25	21	28	26	31	31	34	36	37
3-5	8	20	13	23	18	26	23	29	28	32	33	35	38	38
4-6	9	22	14	25	19	28	24	31	29	34	34	37	39	40
5-6	11	23	16	26	21	29	26	32	31	35	36	38	41	41
6-7	13	25	18	28	23	31	28	34	33	37	38	40	43	43

Figure 14. ESS and LSS for the Activities in R.

Hence, there is a change, and Step 4 does not apply in this case.

Successors of Activity 2-4: Activity 4-6,

Computation of ESS for Activity 4-6 if Activity 2-4 were scheduled equal to BS:

	Cycle						
	1	2	3	4	5	6	7
ESS	21	24	27	30	33	36	39

Hence, there is a change, and Step 4 does not apply in this case.

7. Computation of the schedule given by the forward algorithm (FS) for the activities in M (Activity 6-7):

Activity	Cycle						
	1	2	3	4	5	6	7
6-7	25	28	31	34	37	40	43

Activity 6-7 has the schedule given by the forward algorithm equal to its LSS in all cycles.

8. Scheduling of Activity 6-7 equal to FS (third step of the procedure), Activity 6-7 leaves M,

Activities 4-6 and 5-6 enter M.

9. Computation of the schedule given by the forward algorithm (FS) for the activities in M (Activities 4-6 and 5-6):

Activity	Cycle						
	1	2	3	4	5	6	7
4-6	22	25	28	31	34	37	40
5-6	23	25	27	29	31	38	41

Hence, only Activity 4-6 has the schedule given by the forward algorithm equal to its LSS in all cycles.

10. Scheduling of Activity 4-6 equal to FS (third step of the procedure),

Activity 4-6 leaves M,

Activity 2-4 leaves L and enters K.

11. Verification for every activity in M (Activity 5-6) if the schedule given by the forward algorithm (FS) will not cause a change in the LSS of any predecessor of the activity.

Predecessors of Activity 5-6: Activity 3-5,

Computation of LSS for Activity 3-5 if Activity 5-6 were scheduled equal to FS:

	Cycle						
	1	2	3	4	5	6	7
LSS	16	19	22	25	28	35	38

Hence, there is a change, and Step 5 does not apply in this case.

12. Activities in K: Activity 2-4,

Computation of the schedule given by the backward algorithm (BS) for Activity 2-4 (computed in 5),

Scheduling of Activity 2-4 equal to BS (sixth step of the procedure),

Activity 2-4 leaves K.

13. Computation of ICL.

Apparent lower bound in the number of interruptions for Activities 3-5 and 5-6 before the scheduling of the activities in L (Activity 2-3) equal to BS:

Computation of the schedule given by the backward algorithm (BS) for Activities 3-5 and 5-6:

Activity	Cycle						
	1	2	3	4	5	6	7
3-5	20	23	26	29	32	35	38
5-6	17	19	21	32	34	36	41

Hence, $b_{3-5} = 0$

$b_{5-6} = 2$

(Obs.: The schedule given by the forward algorithm (FS) could also be used for the computation of b_{i-j}).

Apparent lower bound in the number of interruptions for Activities 3-5 and 5-6 after the scheduling of the activities in L (Activity 2-3) equal to BS:

Computation of ESS, after the scheduling of the activities in L = BS, for Activities 3-5 and 5-6:

Activity	Cycle													
	1		2		3		4		5		6		7	
	ESS	LSS	ESS	LSS	ESS	LSS	ESS	LSS	ESS	LSS	ESS	LSS	ESS	LSS
3-5	14	20	17	23	20	26	29	29	32	32	35	35	38	38
5-6	17	23	20	26	23	29	32	32	35	35	38	38	41	41

Computation of the schedule given by the backward algorithm (BS) for Activities 3-5 and 5-6:

Activity	Cycle						
	1	2	3	4	5	6	7
3-5	20	23	26	29	32	35	38
5-6	19	21	23	32	35	38	41

Hence, $a_{3-5} = 0$

$a_{5-6} = 4$

Therefore, $ICL = (a_{3-5} - b_{3-5})c_{3-5} + (a_{5-6} - b_{5-6})c_{5-6}$

$= (0)2 + (2)4 = 8$

14. Computation of ICM.

Apparent lower bound in the number of interruptions for Activities 2-3 and 3-5 before the scheduling of the activities in M (Activity 5-6) equal to FS:

Computation of the schedule given by the backward algorithm (BS) for Activities 2-3 and 3-5:

Activity	Cycle						
	1	2	3	4	5	6	7
2-3	12	14	16	27	29	31	36
3-5	20	23	26	29	32	35	38

Hence, $b_{2-3} = 2$

$b_{3-5} = 0$

Apparent lower bound in the number of interruptions for Activities 2-3 and 3-5 after the scheduling of the activities in M (Activity 5-6) equal to FS:

Computation of LSS, after the scheduling of the activities in M = FS, for Activities 2-3 and 3-5:

Activity	Cycle													
	1		2		3		4		5		6		7	
	ESS	LSS	ESS	LSS	ESS	LSS	ESS	LSS	ESS	LSS	ESS	LSS	ESS	LSS
2-3	6	14	11	17	16	20	21	23	26	26	31	31	36	36
3-5	8	16	13	19	18	22	23	25	28	28	33	35	38	38

Computation of the schedule given by the backward algorithm (BS) for Activities 2-3 and 3-5:

Activity	Cycle						
	1	2	3	4	5	6	7
2-3	6	17	19	21	26	31	36
3-5	16	19	22	25	28	35	38

Hence, $a_{2-3} = 4$

$a_{3-5} = 1$

Therefore, $ICM = (a_{2-3} - b_{2-3})c_{2-3} + (a_{3-5} - b_{3-5})c_{3-5}$

$= (2)1 + (1)2 = 4$

15. ICL is greater than ICM.

Scheduling of Activity 5-6 equal to FS (seventh step of the procedure),

Activity 5-6 leaves M,

Activity 3-5 enters M,

The ESS and LSS for the activities in R are as in 14.

16. Computation of the schedule given by the forward algorithm (FS) for the activities in M (Activity 3-5):

Activity	Cycle						
	1	2	3	4	5	6	7
3-5	16	19	22	25	28	35	38

Activity 3-5 has the schedule given by the forward algorithm equal to its LSS in all cycles.

17. Scheduling of Activity 3-5 equal to FS (third step of the procedure),

Activity 3-5 leaves M,

Activity 2-3 leaves L and enters K.

18. Activities in K: Activity 2-3,

Computation of the schedule given by the backward algorithm (BS) for Activity 2-3 (computed in 14),

Scheduling of Activity 2-3 equal to BS (sixth step of the procedure),

Activity 2-3 leaves K.

19. Set R is empty,

The procedure terminates.

Discussion of Results

As a measure of the effectiveness of the procedure, the total cost of interruptions in the schedule yielded by its use is given below. This cost is compared with the apparent lower bound in the total cost of interruptions and the apparent upper bound in the total cost of interruptions in the project.

Activity	Cost of One Interruption	Ap. Lower Bound in the Cost of Inter.	Cost Yielded by Procedure	Ap. Upper Bound in the Cost of Inter.
1-2	1	0	0	0
2-3	1	2	4	6
2-4	3	0	0	18
3-5	2	0	2	12
4-6	2	0	0	12
5-6	4	8	8	24
6-7	2	0	0	12
	Total	10	14	84

This example is the Project No. 1 in Figure 10.

Example 2:

This is an eight-cycle project with precedence relationships as shown in Figure 15. The number immediately above the arrows represents the duration and the number immediately below the arrow represents the cost of one interruption of the respective activity.

The schedule yielded by the use of the procedure is given in Figure 16.

Discussion of Results

In this case, the seventh step of the procedure is not used, and therefore, the total cost of interruptions in the schedule yielded by the use of the procedure is equal to the apparent lower bound in the total cost of interruptions in the project.

This example is the Project No. 2 in Figure 10.

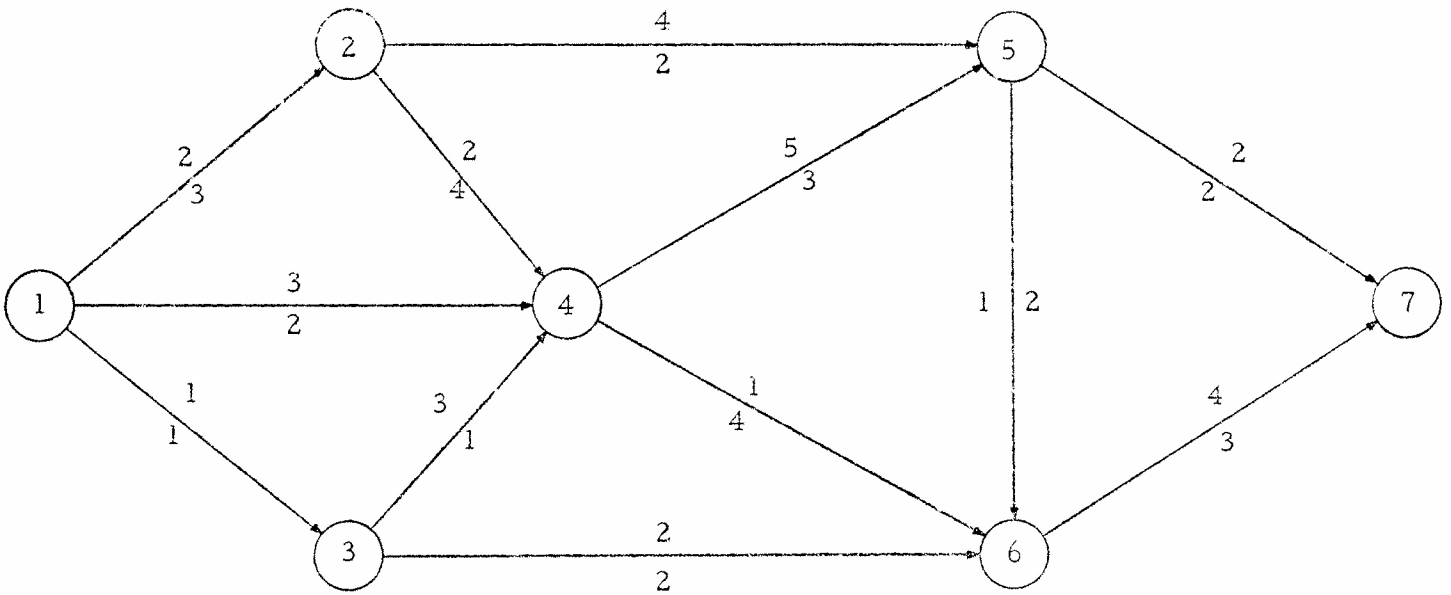
Example 3:

This is a five-cycle project with precedence relationships as shown in Figure 17. The number immediately above the arrows represents the duration, and the number immediately below the arrows represents the cost of one interruption of the respective activity.

The schedule yielded by the use of the procedure is given in Figure 18.

Discussion of Results

This is the largest project that was scheduled by the procedure.



Number immediately above the arrow = duration of the activity

Number immediately below the arrow = cost on one interruption of the activity

Figure 15. Network Representation of One Cycle of the Project of Example 2.

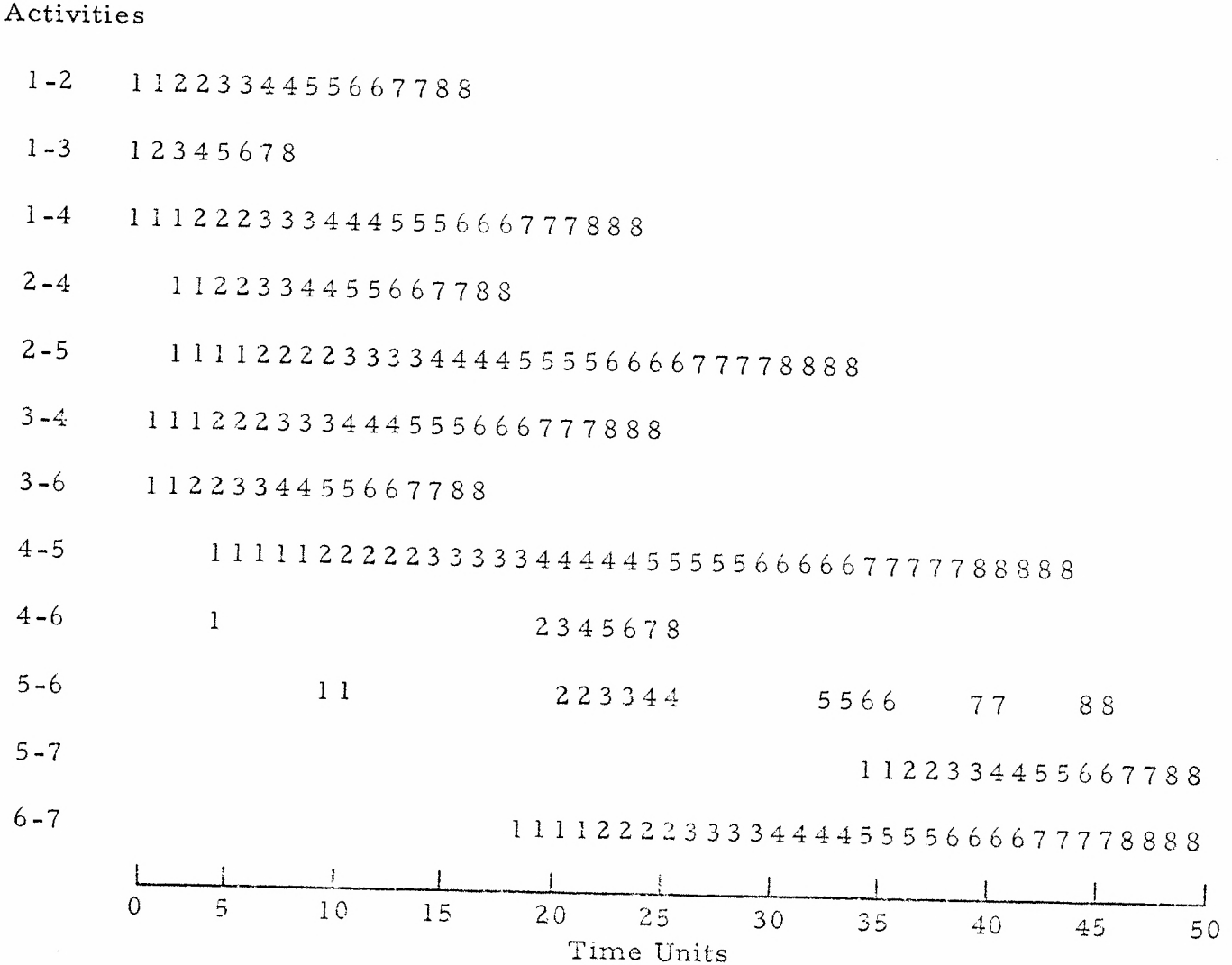
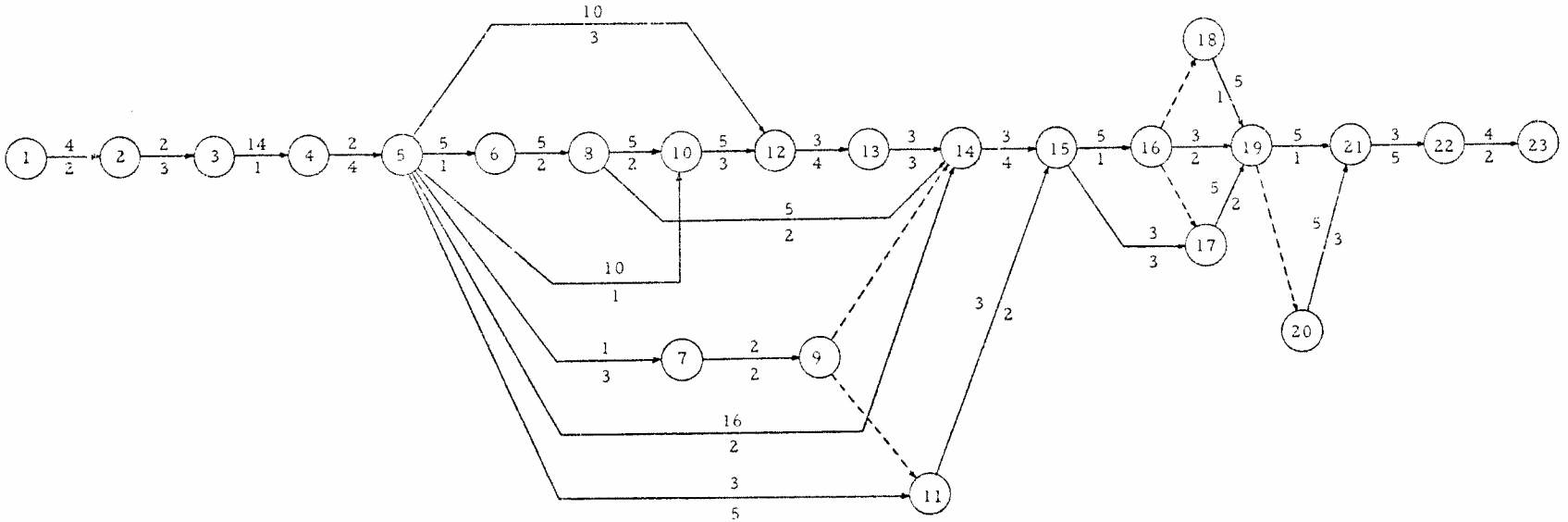


Figure 16. A Schedule for the Cyclic Project of Example 2.



Number immediately above the arrow = duration of the activity

Number immediately below the arrow = cost of one interruption of the activity

Figure 17. Network Representation of One Cycle of the Project of Example 3.

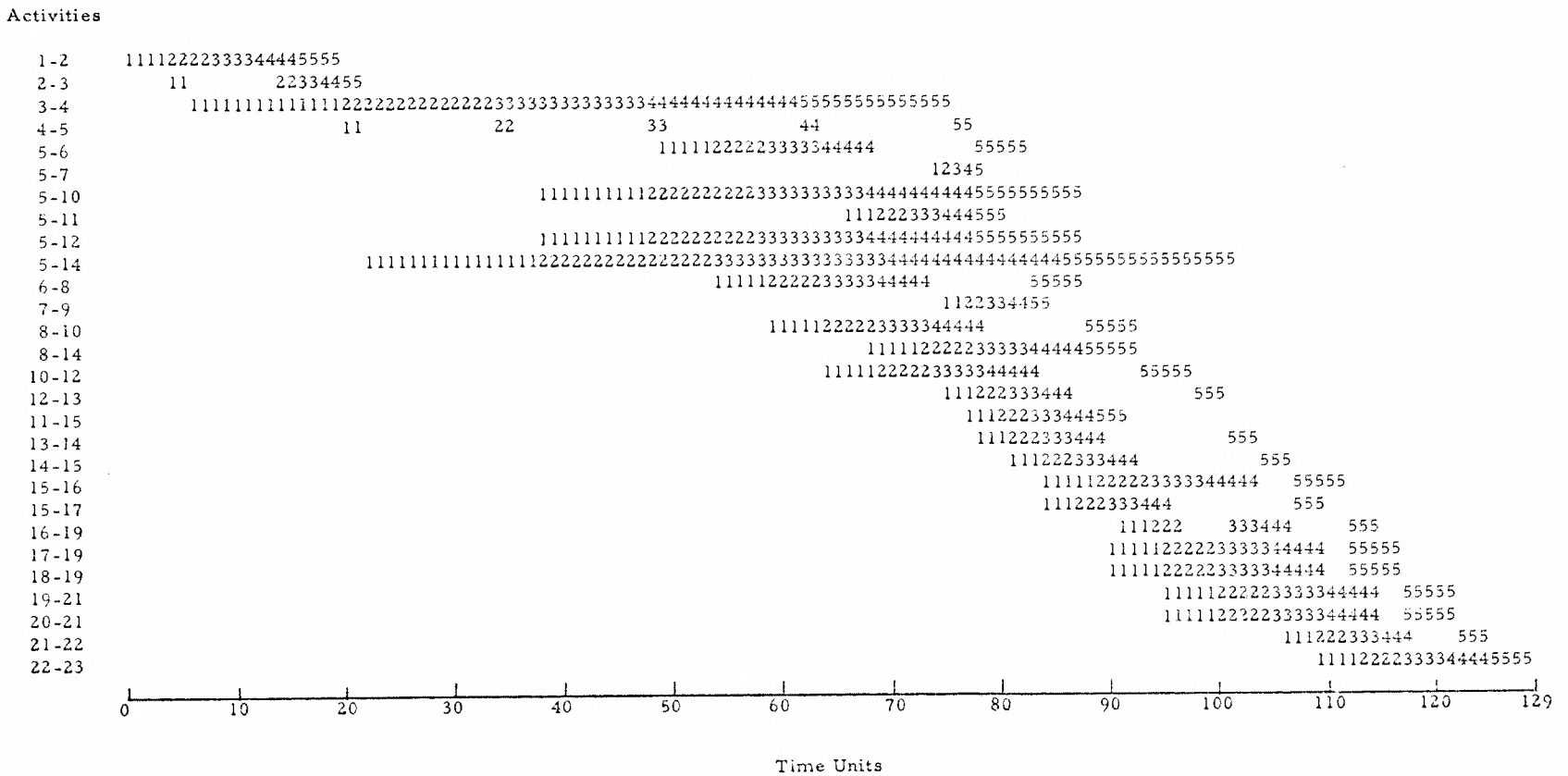


Figure 18. A Schedule for the Cyclic Project of Example 3

The number of activities and cycles is approximately what one might encounter on an apartment or office building project. The manual computation required approximately fifty man-hours. The savings in interruption costs on an actual project would only have to be approximately $300 for the procedure to be economically justifiable. As the schedule produced is not an intuitively obvious one, it is entirely reasonable to expect that an intuitively generated schedule would have had more than $300 in excessive interruption costs.

APPENDIX C

FLOWCHART OF THE OVERALL PROCEDURE

APPENDIX C

FLOWCHART OF THE OVERALL PROCEDURE

Notation:

ESS: earliest time the start of an activity can be scheduled in a cycle and still be within the critical path requirements,

LSS: latest time the start of an activity can be scheduled in a cycle and still be within the critical path requirements,

R: set of all activities not completely scheduled,

K: a subset of R formed by the activities which have all predecessors and all successors completely scheduled,

L: a subset of R formed by the activities which have all predecessors completely scheduled, but not all successors,

M: a subset of R formed by the activities which have all successors completely scheduled, but not all predecessors,

BS: schedule given by the "backward algorithm,"

FS: schedule given by the "forward algorithm,"

ICL: increase in the apparent lower bound in the total cost of interruptions for the remaining activities to be scheduled if the activities in L" are scheduled equal to BS,

ICM: increase in the apparent lower bound in the total cost of interrup-

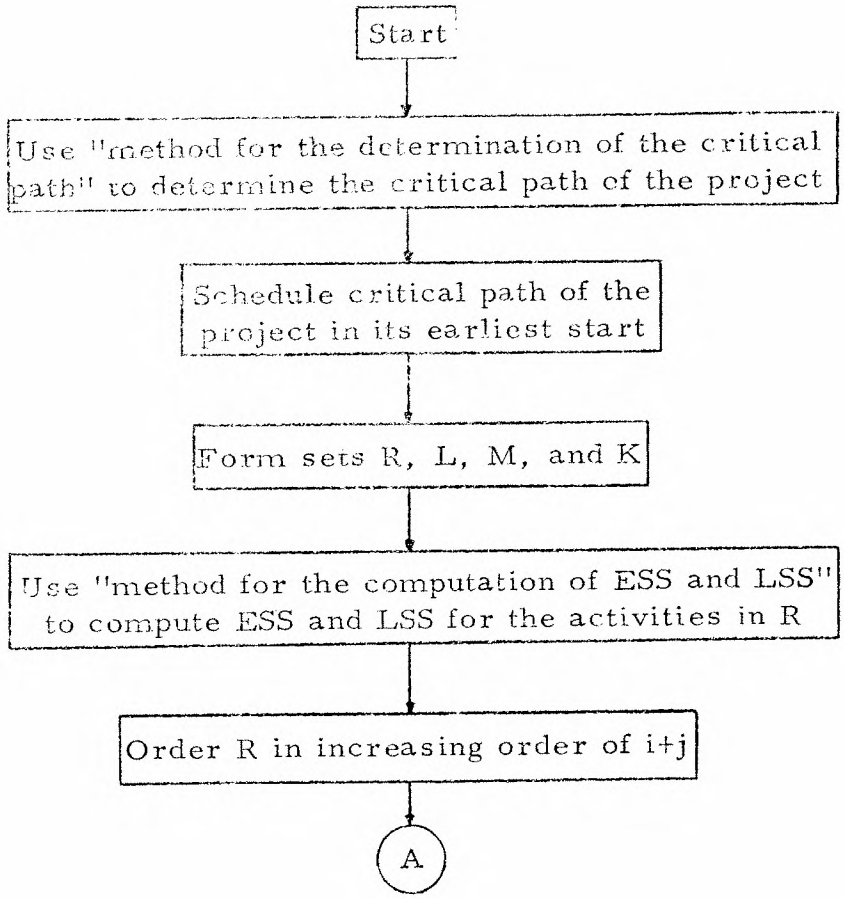
tions for the remaining activities to be scheduled if the activities in M'' are scheduled equal to FS,

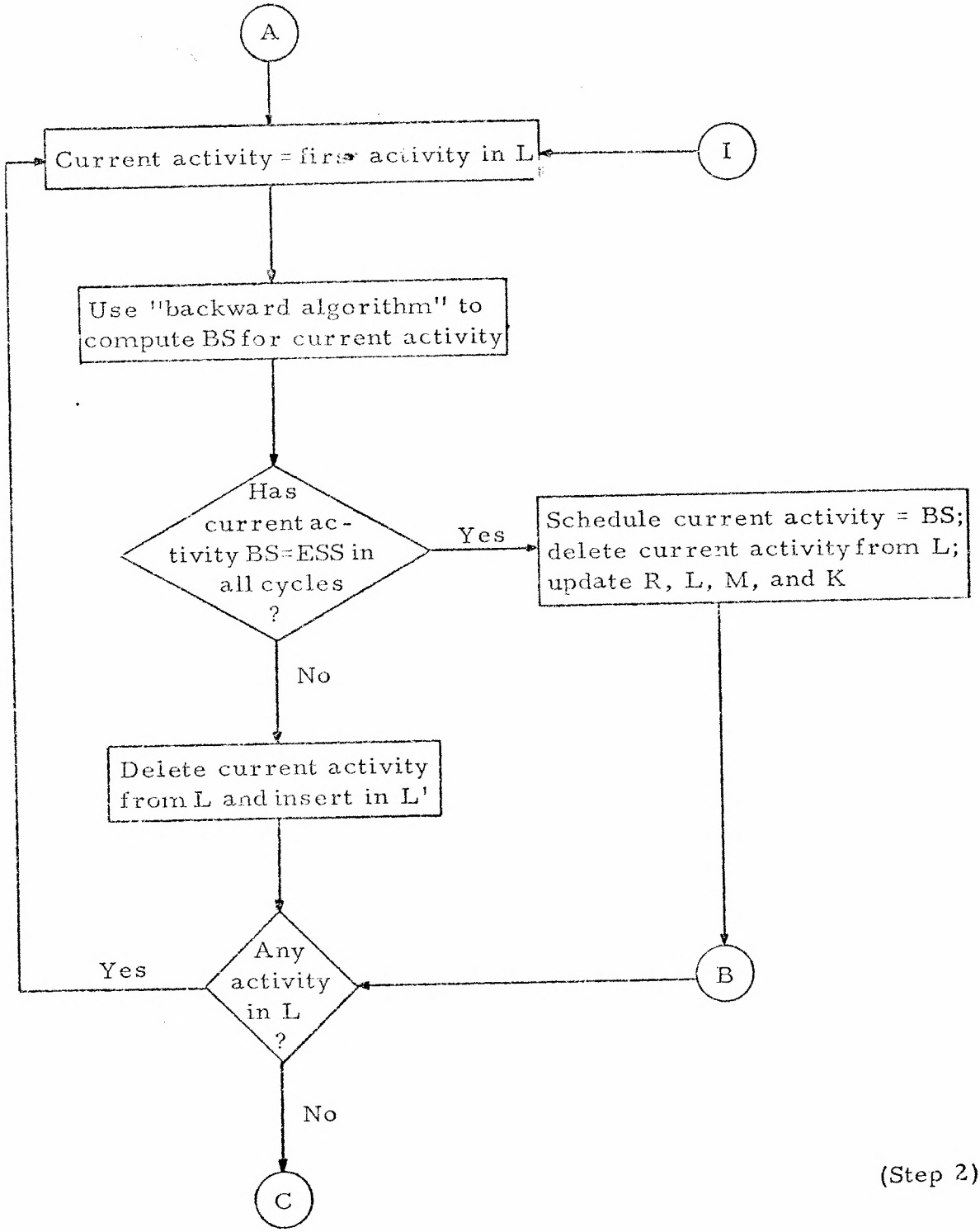
L': set of activities in L examined for BS equal to ESS,

L'': set of activities in L examined for BS equal to ESS, and examined for change in ESS of any successor if the activities are scheduled equal to BS,

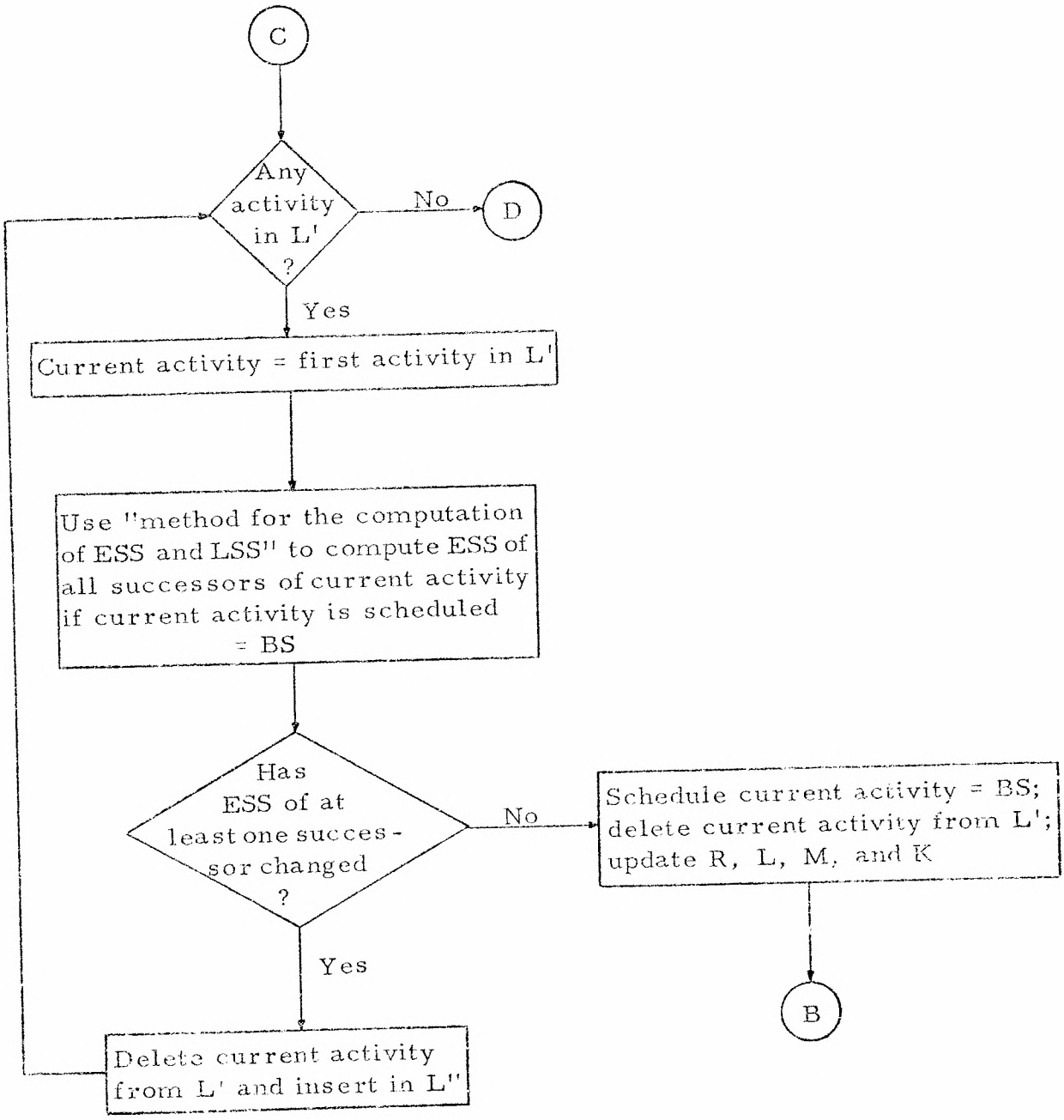
M': set of activities in M examined for FS equal to LSS,

M'': set of activities in M examined for FS equal to LSS, and examined for change in LSS of any successor if the activities are scheduled equal to FS.

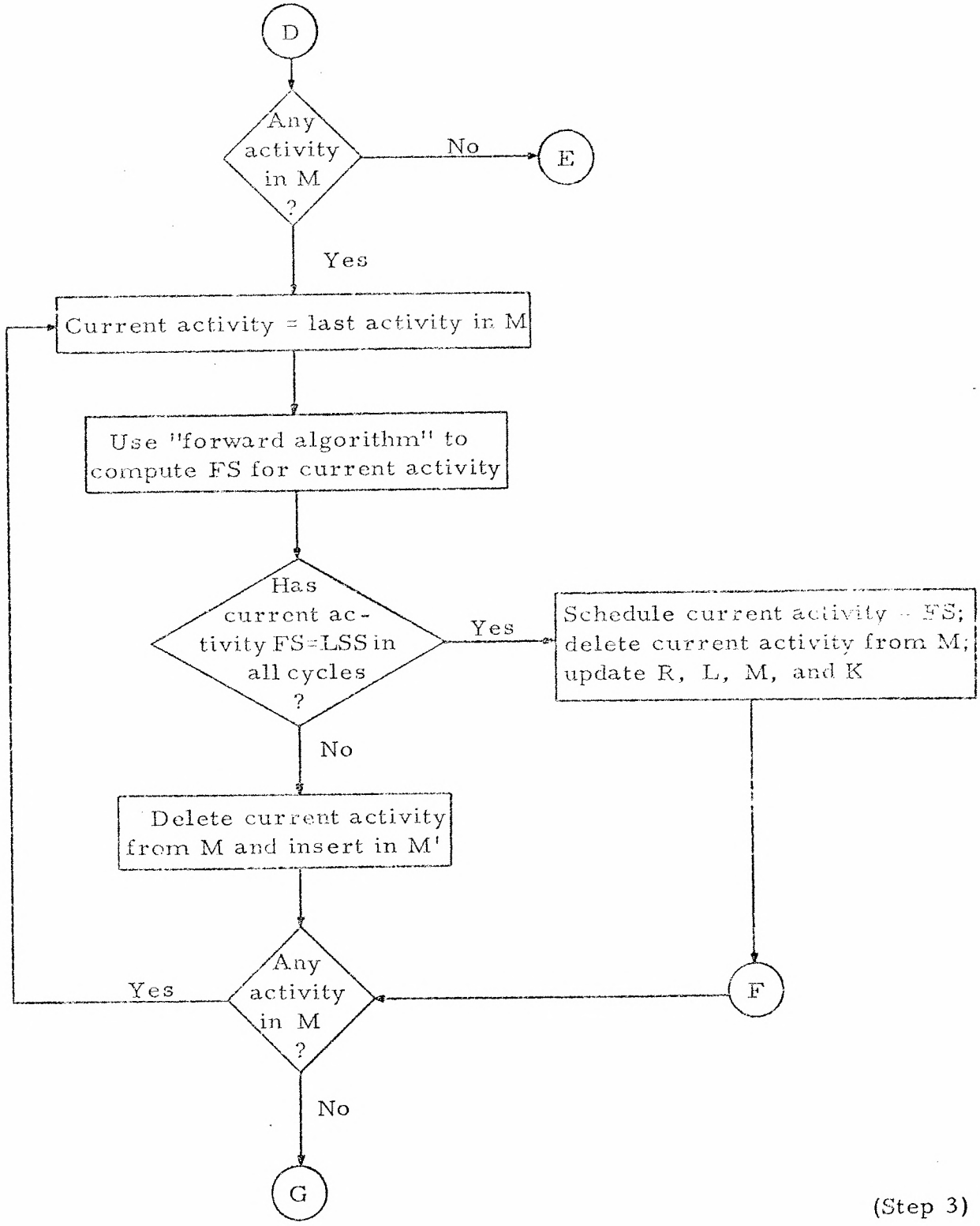



A
Current activity = first activity in L
I
Use "backward algorithm" to compute BS for current activity
Has current activity BS=ESS in all cycles ?
Yes
Schedule current activity = BS; delete current activity from L; update R, L, M, and K
No
Delete current activity from L and insert in L'
Any activity in L ?
Yes
B
No
C

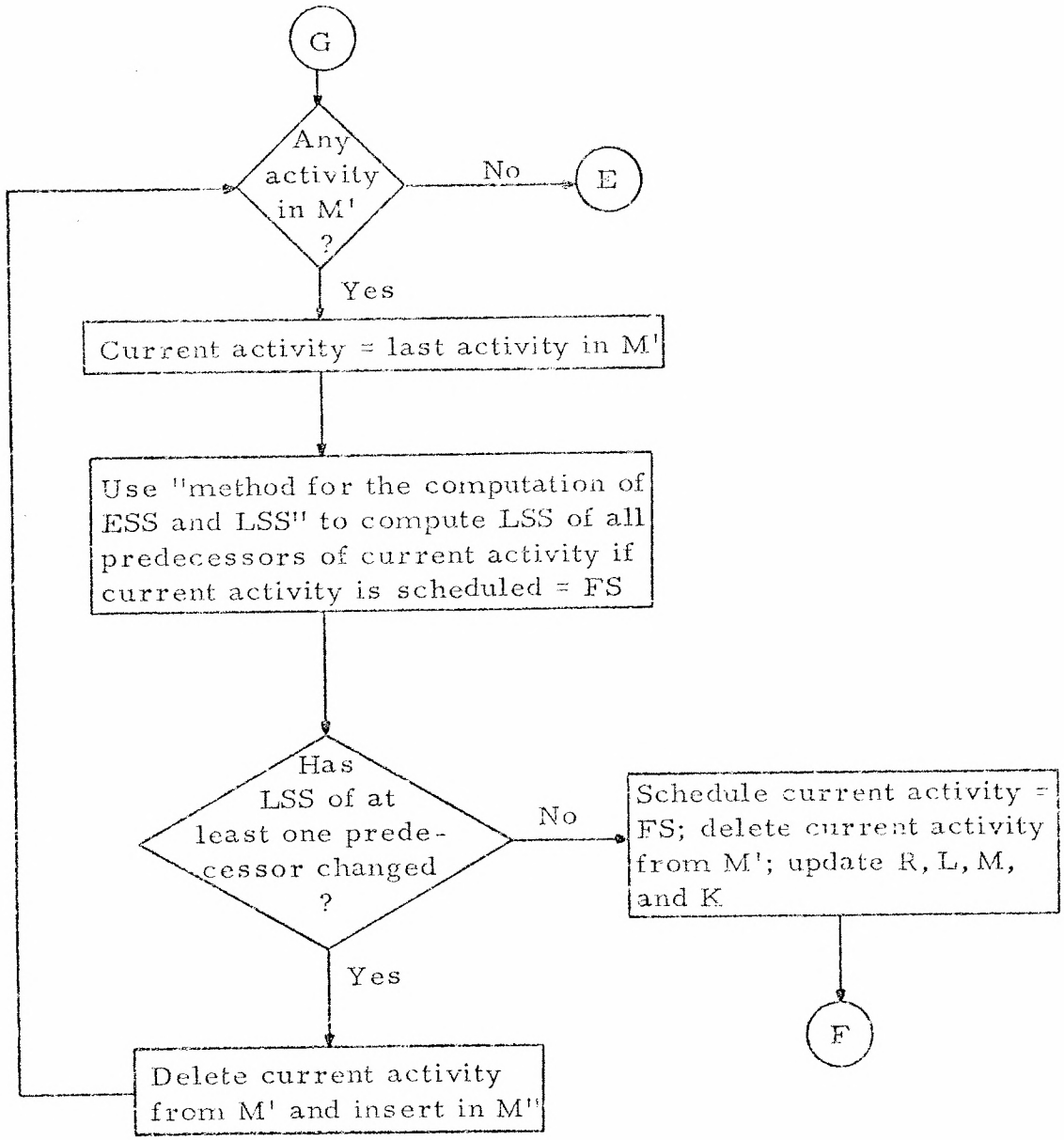
(Step 2)


C
Any activity in L' ?
No
D
Yes
Current activity = first activity in L'
Use "method for the computation of ESS and LSS" to compute ESS of all successors of current activity if current activity is scheduled = BS
Has ESS of at least one succes-sor changed ?
No
Schedule current activity = BS; delete current activity from L'; update R, L, M, and K
B
Yes
Delete current activity from L' and insert in L''

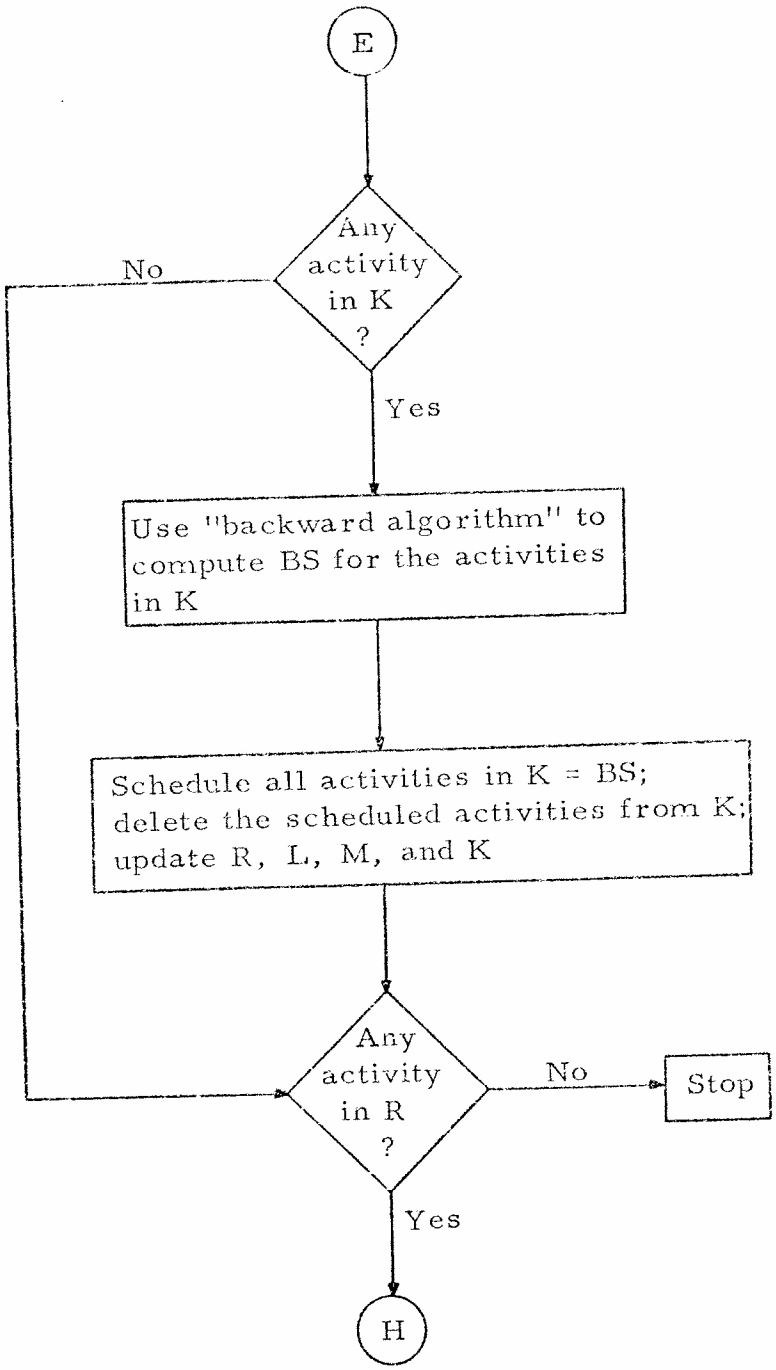
(Step 4)


D
Any activity in M ?
No
E
Yes
Current activity = last activity in M
Use "forward algorithm" to compute FS for current activity
Has current activity FS=LSS in all cycles ?
Yes
Schedule current activity = FS; delete current activity from M; update R, L, M, and K
No
Delete current activity from M and insert in M'
Any activity in M ?
Yes
F
No
G

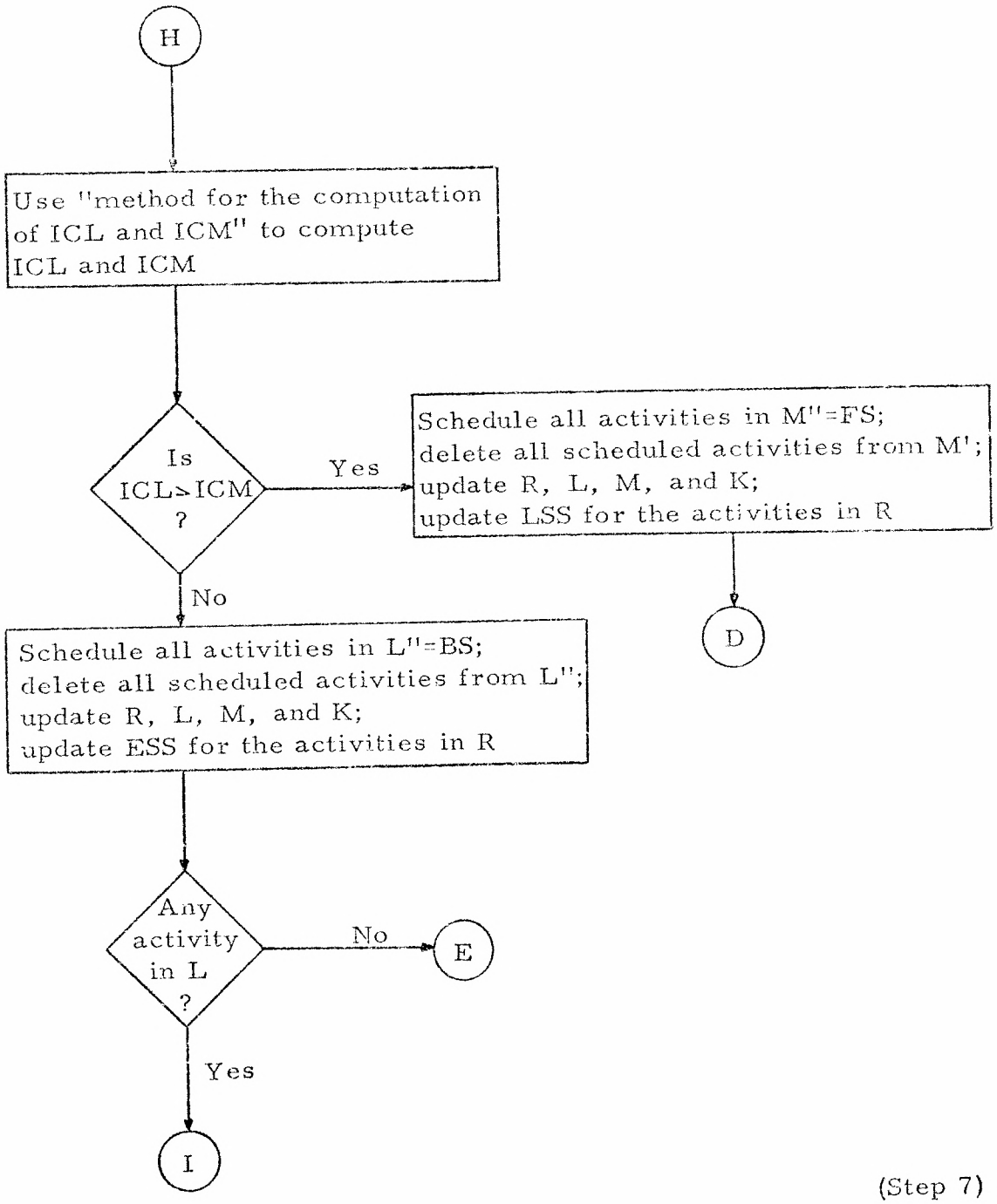
(Step 3)


G
Any activity in M' ?
No
E
Yes
Current activity = last activity in M'
Use "method for the computation of ESS and LSS" to compute LSS of all predecessors of current activity if current activity is scheduled = FS
Has LSS of at least one predecessor changed ?
No
Schedule current activity = FS; delete current activity from M'; update R, L, M, and K
F
Yes
Delete current activity from M' and insert in M''

(Step 5)


E
Any activity in K ?
No
Yes
Use "backward algorithm" to compute BS for the activities in K
Schedule all activities in K = BS; delete the scheduled activities from K; update R, L, M, and K
Any activity in R ?
No
Stop
Yes
H

(Step 6)


H
Use "method for the computation of ICL and ICM" to compute ICL and ICM
Is ICL≥ICM ?
Yes
Schedule all activities in M"=FS; delete all scheduled activities from M'; update R, L, M, and K; update LSS for the activities in R
D
No
Schedule all activities in L"=BS; delete all scheduled activities from L"; update R, L, M, and K; update ESS for the activities in R
Any activity in L ?
No
E
Yes
I

(Step 7)

BIBLIOGRAPHY

1. Battersby, A., Network Analysis for Planning and Scheduling, Macmillan & Company, Ltd., London, 1967, 414 pages.

2. Burgess, A. R., and Killebrew, J. B., "Variation in Activity Level on a Cyclical Arrow Diagram," Journal of Industrial Engineering, Vol. 13, No. 2 (1962), pp. 76-83.

3. Burney, S. M., Jr., A Technique for Scheduling Subcontracted Work in a Cyclic Project, M.S. Thesis, Georgia Institute of Technology, Atlanta, Georgia, 1971.

4. Fisher, C., and Nemhauser, G. L., "Multicycle Project Planning," Journal of Industrial Engineering, Vol. 18, No. 4 (1967), pp. 278-284.

5. Moder, J. J., and Phillips, C. R., Project Management with CPM and PERT, Second Edition, Van Nostrand Reinhold Company, New York, 1970, 360 pages.

6. Wiest, J. D., and Levy, F. K., A Management Guide to PERT/CPM, Prentice-Hall, Inc., New Jersey, 1969, 170 pages.

Self-Organizing Production Systems

A THESIS
Presented to
The Academic Faculty

by

Donald D. Eisenstein

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in Industrial and Systems Engineering

Georgia Institute of Technology
August 20, 1992

Self-Organizing Production Systems

Approved:

John J. Bartholdi, III, Chairman, Co-Advisor

H. Donald Ratliff, Co-Advisor

Leonid Bunimovich

Robert D. Foley

Charlotte Jacobs-Blecha

Date approved by chairman 8/12/92

In memory
of my grandfather,
Harry Herman

Acknowledgements

There are many people to thank, who have offered their friendship, encouragement, and technical expertise. I must start with my thesis advisors, John Bartholdi and Don Ratliff. I could not exaggerate the role John has played as my advisor. The technical expertise alone he provided was more than substantial, but in addition, John has become a friend and inspiration. His unselfishness borders on insanity — thanks John, for everything. I started my education at Georgia Tech working with Don as a research assistant and he was the one I called when I was ready to return to complete my Ph.D. He welcomed me back with open arms. Don has always had the right question to ask; I was lucky enough to be in his office when he asked one about production lines.

In addition I had an unbelievable resource of faculty at Georgia Tech from which to tap. I thank Lyonia Bunimovich and Bob Foley, both on my thesis committee, for their help and guidance on this research. I thank Charlotte Jacobs-Blecha, also on my thesis committee, for her enthusiasm, friendship and suggestion that we look into the apparel industry. I must also thank John Jarvis, who along with Don Ratliff, guided my research on my first visit to Tech. In addition I would like to thank Steve Hackman, George Nemhauser, Gary Parker, Craig Tovey, and John Vande Vate for their help and encouragement.

I must also mention the faculty and friends from Southern Methodist University. Foremost is Jeff Kennington, who suggested I look into his own alma mater for graduate studies. Also Jay Aronson, who spent considerable time with me and fostered my interest in Operations Research. Finally I thank Jose Dula and Betty Hickman for their friendship and support.

If there was one advantage in making two visits to Georgia Tech, it must be that I have had the pleasure of meeting two groups of students. From the old gang I thank Mike Trick for his friendship and advise over the years. I don't think I would have returned to Tech and completed my Ph.D. without it — thanks Mike. I must also thank Ananth Iyer and Bill Nulty from the old gang (Bill plays an awesome game of Millipede). From the new gang I must thank Carolyn Wingfield, whose support and friendship have made all the difference down the home stretch. I must also thank Anuj Mehrotra with whom I have shared many laughs, beers and falafels. I also thank Samir Amiouny, Betsy Bourn, Ismael de Farias, Kevin Gue, Chyi-Fu Hong, Sophie Lapierre, Antonio Moretti, Kevin McCroan, Andy Sandifer, Bryan Stutzman, Pam Vance, and Linda Whitaker for their friendship and adventures.

In addition, I would like to show my appreciation to the staff at Georgia Tech, they were always willing to help whenever I needed something. I single out our computer wizard Richard Robison, who was always there with equal patience for both technical and not so technical problems.

I must mention David Jablinski, who has been my friend for longer than I can remember (although he actually remembers where I sat in first grade!). We went from elementary school through undergraduate studies together, he has always been a confidant and inspiration — thanks Dave.

I thank by parents, Paul and Selma, for the love and support they continue to provide. (Mom, you said you wanted me to be a doctor, although I'm not quite sure this is what you had in mind!)

Finally I thank Darla, my closest friend, for her love and patience over the years — this effort is as much hers as mine.

Contents

Summary

We describe a production line used in the apparel industry in which n workers move among m stations. Each worker independently follows a simple rule that determines what to do next. We model the production line as a discrete dynamical system. We show that if the workers have essentially the same skill, that the production rate will converge to the largest possible rate. If workers differ in skill then we prove that, if the workers are ordered from slowest to fastest, then independently of the stations at which they begin, a stable allocation of work will spontaneously emerge. At this stable allocation each worker repeats the same interval of the production line and the production rate is constant. Furthermore, even if stations have different amounts of work, then—as long as no station has "too much"—in the emergent allocation all workers contribute the same clock time to the production of each item. If workers that differ in skill are not ordered from slowest to fastest we observe that the dynamics of the system exhibit complicated and unpredictable behavior.

CHAPTER 1

Introduction

Traditional means of organizing a production line, such as a classical assembly line, are inflexible. In a classical assembly line, workers are assigned fixed work stations and the station with the greatest work content determines the production rate. Realistically, there are only two ways to change the production rate: either change the number of shifts, or else redistribute the tasks, tools, and parts over different stations. The first allows only coarse adjustments and the second is expensive and disruptive. This inflexibility is partly due to the way in which traditional production is organized, where a centralized authority designs a globally coördinated pattern of material movement and task execution that is then rigidly followed by all workers.

It is particularly important that production systems be flexible when products have extreme seasonalities or short life-cycles, such as in the apparel industry. To increase flexibility of production, there has recently been introduced into the apparel industry a variation of the assembly line in which there are fewer workers than stations and workers walk to adjacent stations to continue work on an item. Control of the line is decentralized: Each worker independently follows a simple rule that determines what to do next. This idea has been commercialized by Aisin Seiki Co.,

Ltd., a subsidiary of Toyota, and named the "Toyota Sewn Products Management System", or TSS[1]. It is marketed in the western hemisphere by TSS Americas, Inc. According to advertising literature it has been used successfully in the manufacture of many types of sewn products, including furniture, shoes, and fish nets.

We will show that when TSS workers are of the same skill then the production rate of the line will converge to the largest possible rate. If workers are of differing skills then we shall prove that, when TSS workers are sequenced from slowest to fastest, then, during the natural operation of the line, the work content of the product will be spontaneously reallocated among the workers to balance the line—without conscious intention by the workers or intervention by management. This capacity for self-organization allows management to fine-tune the production rate by simply changing the number of workers on the line, which in turn elicits a spontaneous reallocation of work.

We furthermore observe that such self-organizing behavior fails to occur if workers are not sequenced from slowest to fastest. That in general, the behavior of such lines is complex and unpredictable.

[1]A registered trademark of Aisin Seiki Co., Ltd.

CHAPTER 2

A model of TSS

Call each instance of the product an *item* and consider a flow line in which each item requires identical processing on the same sequence of m work stations, as in Figure 2.1. A station can process at most one item at a time, and exactly one worker is required to accomplish the processing. (This type of line is typical of the apparel industry, where each work station is generally a sophisticated sewing machine through which an item must be guided by a single worker.)

We assume that each item requires the same total amount of processing according to some work standard, and we normalize that total to one "time unit". Let the processing requirement at station j be p_j, a fixed percentage of the total standard work content of the product.

Figure 2.1: A simple flow line in which each item requires identical processing on the same sequence of work stations.

The TSS line functions as a sort of "bucket brigade" in which n workers move down the line, each devoted to a single item. When the last worker completes her item—all the workers we have seen are female—she relinquishes it, moves back up the line, and takes over the work of her predecessor, who in turn preëmpts her predecessor, and so on until the first worker, after having been preëmpted, introduces a new item to the line[1]. This behavior can be realized by requiring each worker to independently follow this rule:

TSS Rule (forward part) Remain devoted to a single item, and process it on successive work stations, queueing before a busy work station if necessary. If you complete processing the item(if you are the last worker) or if you are preëmpted by another worker, then relinquish the item and begin to follow the Backward Part.

TSS Rule (backward part) Walk back toward the beginning of the line until you encounter an item. If you are the first worker retrieve a new item from the input buffer, else take over the item of your predecessor. Begin following the Forward Part.

(In fact none of the TSS lines we have observed follow this rule exactly. Instead, there are usually additional restrictions on the behavior of some workers, such as "worker 2 should never proceed past station 5". The TSS "rule" is our abstraction, which is sufficient to generate the essential behavior of TSS lines.)

A typical TSS line that we observed was devoted to the production of ladies slacks. It had seven stations and was staffed by three workers. The total work content of the slacks was about seven minutes and the work at each station was 45–90 seconds. It required 5–7 seconds after the completion of an item for all the workers to walk back and take over the work of their predecessors.

[1]Note that it is the worker who is preëmpted and not the item.

Figure 2.2: The standard work content of the product is represented as a line segment normalized to length 1, which is partitioned into intervals corresponding to the work stations. The position of worker i is given by x_i, the cumulative fraction of work content completed on her item.

How might one describe the behavior of a TSS line? It is difficult to visualize the movements of workers on a TSS line because they operate asynchronously. The line may be viewed as a dynamical system (Devaney, 1989) by expressing the position of worker i as the fraction $x_i(t)$ of work completed on her item by time t, as illustrated in Figure 2.2; then the state of the system at time t is given by the vector of worker positions $\mathbf{x}(t) = (x_1(t), \ldots, x_n(t))$. The phase space of the system is a subset of the unit hypercube, as illustrated in Figure 2.3, where $x_1 \leq x_2 \leq \cdots \leq x_n$ because the TSS rule does not allow workers to pass one another. The saw-toothed edge of the phase space arises because no more than one worker can use a station at a time.

Unlike most other production models, ours includes specific representation of the human workers. We model each worker by a velocity function $v_i(x)$ that gives her instantaneous "work velocity" at position $x \in [0, 1]$. The only restrictions we place on the function v_i are that it be continuous almost everywhere and bounded in the following sense: there exist numbers b and B such that $0 < b \leq v_i(x) \leq B$ for all $x \in [0, 1]$. Our model includes, for example, the very natural one in which the work to produce an item is comprised of many primitive task elements, and each worker's velocity is constant (and nonzero) over any particular task element. Indeed, this is the model used in the apparel industry, where each worker has a documented skill profile giving her velocity on different tasks as a percentage of work standards.

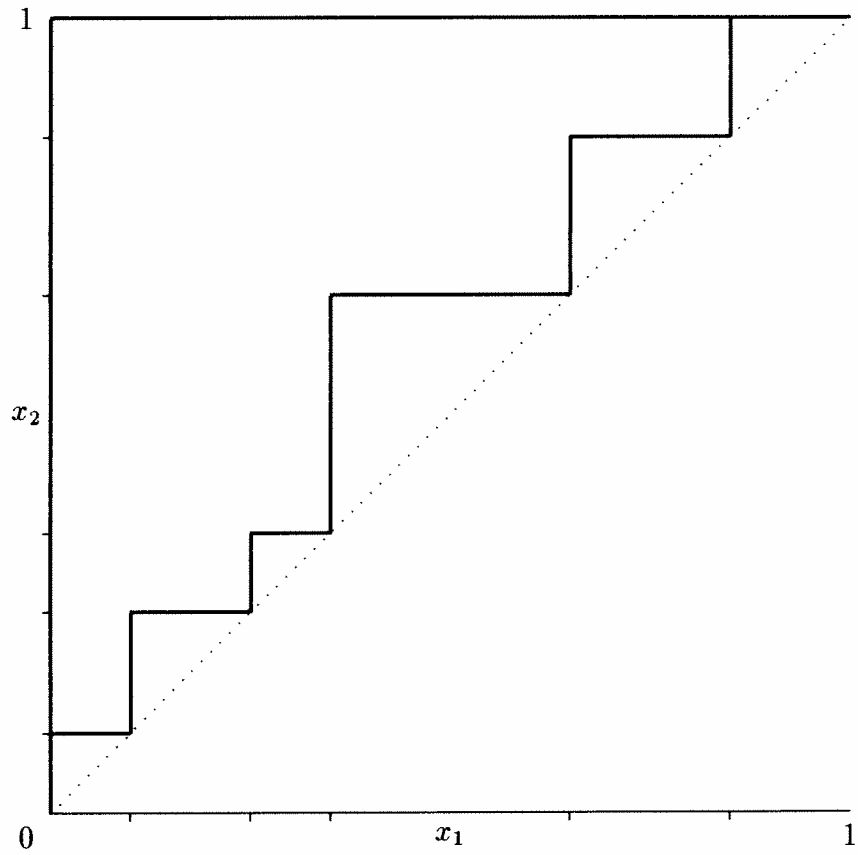


Figure 2.3: The phase space of a TSS line with two workers, whose positions are given by (x_1, x_2), is that portion of the upper triangle outlined in bold. The tick marks on the axes correspond to the partition of work among the stations.

We define $t_i(x, x')$ to be the clock time required for worker i to move without interruption from position x to position x', where $x \leq x'$ and $x, x' \in [0, 1]$. By the properties of v_i, the function $t_i(x, x')$ is well-defined and may be expressed as follows:

$$t_i(x, x') = \int_x^{x'} \frac{dz}{v_i(z)}.$$

Note that $t_i(x, x')$ is strictly increasing in x' and strictly decreasing in x.

Now we make a modelling assumption that is consistent with the TSS lines we have seen: that the time to walk back and preëmpt a worker is small. Therefore we can imagine that when the last worker finishes an item, then—at the same instant—worker n preëmpts worker $n-1$, who preëmpts worker $n-2$, ..., who preëmpts worker 1, who introduces a new item into the system. We say that the line *resets* at such an instant. This simplification frees us from worry about the details of the continuous-time TSS dynamics function $\mathbf{x}(t)$; instead we can restrict our attention to the sequence $\{\mathbf{x}^0, \mathbf{x}^1, \mathbf{x}^2, \ldots\}$ of worker positions at those instants when the line resets[2]. Continuing the metaphor of a dynamical system, we call each $\mathbf{x}^p$ an *iterate* of the TSS system and the sequence of worker positions the *orbit* beginning at $\mathbf{x}^0$. Let f be the function, defined implicitly by the TSS rule, that maps the vector of worker positions after one reset to that after the subsequent reset, so that $\mathbf{x}^{p+1} = f(\mathbf{x}^p)$.

Let $\mathbf{x}^p$ be the pth iterate of a TSS line. Then by definition $x_1^p = 0$ and for convenience we define $x_{n+1}^p = 1$. Also define $p_0 = 0$, and let $P_k = \sum_{j=0}^{k} p_j$ be the cumulative amount of work invested in an item when it has just completed processing on station k. The work at station k corresponds to the disjoint open interval (P_{k-1}, P_k) $(k = 1, \ldots, m)$; and, because workers cannot use the same station

[2]Such a subset of the phase space is called a *Poincaré section*. Restricting the phase space in this manner is a standard technique for analyzing complicated dynamics (Morrison, 1991).

at the same time, no such interval can contain more than one x_i. Sometimes we will need to know the endpoints of the station (interval) containing position x; accordingly we define

$$\underline{x} = P_{k-1} \text{ if } x \in [P_{k-1}, P_k);$$

and

$$\overline{x} = P_k \text{ if } x \in (P_{k-1}, P_k].$$

2.1 Repeatable behavior

A potential problem with (our model of) TSS lines is that there are no restrictions on where a worker might work, so that in principle she must be trained to perform all tasks on the line. Good management sense suggests that, to avoid excessive training costs, a TSS line should be run so that each worker repeatedly executes the same interval of work content. Our first result shows that for any TSS line there exists an initial set of positions, $\mathbf{x}^*$, to which workers, in the absence of perturbations, will always return at resets; that is $\mathbf{x}^* = f(\mathbf{x}^*)$. Therefore, each worker repeats the same interval of work content at each iteration.

Theorem 2.1 *For any TSS line, there exists a fixed point* $\mathbf{x}^* = f(\mathbf{x}^*)$*; that is, there exists worker positions* $\mathbf{x}^*$ *such that if the workers start at positions* $\mathbf{x}^*$*, then they will always reset to* $\mathbf{x}^*$*.*

Proof We extend f so that its domain is the entire closed n-cell defined by $0 \leq x_1 \leq x_2 \leq \cdots \leq x_n \leq 1$. To do this, make the natural extension to the TSS rule so that, if more than one worker has partially completed work at the same station, then the worker at that station with highest index $i = 1, \ldots, n$ has priority and the others must wait to use this station until she has finished. (We emphasize

that this extension is never necessary in practice; it is just a means of proving our theorem.) Let g be the dynamics function corresponding to the extended TSS rule; as we shall show shortly, g is continuous on its domain and therefore, by Brouwer's Fixed Point Theorem, has a fixed point (Bollobás, 1990). Furthermore, this fixed point must lie within the natural domain of f because, by the logic of the extended TSS rule, no point in the extended domain can remain fixed under g. Therefore, since f agrees with g on the domain of f, the fixed point with respect to g must also be a fixed point with respect to f.

Now we show that the extended TSS function g is continuous. To see this, imagine two versions of the TSS line operating in parallel, so that there are two versions of each worker i, one at position x_i^p on one line and the other at nearby position y_i^p on the other line. During the next iteration the two versions of worker i travel forward to positions x_{i+1}^{p+1} and y_{i+1}^{p+1} respectively.

Call the worker at $\min\{x_i^p, y_i^p\}$ the *first version* of worker i and the one at $\max\{x_i^p, y_i^p\}$ the *second version* of worker i.

Now imagine the two versions of the TSS line operating with the following revised protocol: For each worker i, the second version does not begin work until the first version (on the other line) draws abreast; this must occur within time T_i where,

$$T_i = (1/b)\left(\sum_{j=i}^{n}\left|x_j^p - y_j^p\right|\right).$$

Furthermore, from this time onward in the revised protocol the two versions of worker i will behave identically. Therefore the distance between the versions of worker i is non-decreasing, so that

$$\left|x_{i+1}^{p+1} - y_{i+1}^{p+1}\right| \leq \left|x_i^p - y_i^p\right|.$$

Now we claim, under the actual TSS protocol, that unless the line resets first, the first version of worker i will reach the starting position of the second version of i within time T_i. This holds by backwards induction. It is clearly true for $i = n$. Assume it is true for $i = k$ and consider the two versions of worker $k-1$. By the induction hypothesis, after time T_k the first version of worker $k-1$ will not be blocked en route to position $\max\left\{x_{k-1}^p, y_{k-1}^p\right\}$; and the travel time to this position cannot exceed $\left|x_{k-1}^p - y_{k-1}^p\right|/b$. Therefore the total elapsed time to reach this position can never exceed T_{k-1}.

Under the actual TSS protocol, the two versions of worker i might have separated during the time T_i; but this separation cannot exceed the farthest a worker can move within time T_i, which is BT_i. Therefore, under the actual TSS protocol,

$$\left|x_{i+1}^{p+1} - y_{i+1}^{p+1}\right| < BT_i + \left|x_i^p - y_i^p\right|,$$

so that

$$\begin{aligned}
\left|g\left(\mathbf{x}^p\right) - g\left(\mathbf{y}^p\right)\right| &= \left|\mathbf{x}^{p+1} - \mathbf{y}^{p+1}\right| \\
&\leq \sum_{i=1}^{n} \left|x_i^{p+1} - y_i^{p+1}\right| \\
&< nBT_1 + \sum_{i=1}^{n} \left|x_i^p - y_i^p\right| \\
&= \left(1 + \frac{nB}{b}\right)\left(\sum_{i=1}^{n} \left|x_i^p - y_i^p\right|\right).
\end{aligned}$$

Therefore $\left|g\left(\mathbf{x}^p\right) - g\left(\mathbf{y}^p\right)\right|$ is small whenever $\left|\mathbf{x}^p - \mathbf{y}^p\right|$ is sufficiently small, and so g is continuous. □

2.2 Imputed allocations of work

It will be easier to analyze the behavior of a TSS line if we introduce a new coördinate system to keep track not just of the positions of the workers but also of the amount of work time between the positions of successive workers. The vector $\mathbf{x}^p$ can be interpreted as suggesting a partition of the work during the next iteration, with the interval of work content $[x_i^p, x_{i+1}^p]$ assigned to worker i. Roughly speaking, the *allocation* a_i is the clock time required for worker i to complete her suggested share of work. More precisely,

$$a_n^p = t_n\left(x_n^p, 1\right), \tag{2.1}$$

and, if worker $i+1$ is not blocked at the start of the p-th production cycle,

$$a_i^p = t_i\left(x_i^p, x_{i+1}^p\right) + \max\left\{0,\, t_{i+1}\left(x_{i+1}^p, \overline{x_{i+1}^p}\right) - t_i\left(x_i^p, \underline{x_{i+1}^p}\right)\right\}. \tag{2.2}$$

Expression 2.2 consists of two terms: the first is the clock time required for worker i, if she is not blocked en route, to reach the starting position x_{i+1}^p of her successor; and the second term is the delay if worker i is blocked en route, in which case i reaches $i+1$'s station at time $t_i\left(x_i^p, \underline{x_{i+1}^p}\right)$, but $i+1$ does not relinquish that station until time $t_{i+1}\left(x_{i+1}^p, \overline{x_{i+1}^p}\right)$.

If worker $i+1$ is blocked at the start of the production cycle, then we define the allocation of worker i to be

$$a_i^p = \max\left\{t_i\left(x_i^p, x_{i+1}^p\right),\, a_{i+1}^p\right\}. \tag{2.3}$$

Expression 2.3 defines the allocation in "degenerate" cases, when successive workers block each other.

We identify three types of allocations: a_i^p is a *simple* allocation if $a_i^p = t_i\left(x_i^p, x_{i+1}^p\right)$ from expression 2.1, 2.2 or 2.3, a_i^p is a *delay allocation* if $a_i^p = t_i\left(\underline{x_{i+1}^p}, x_{i+1}^p\right) +$

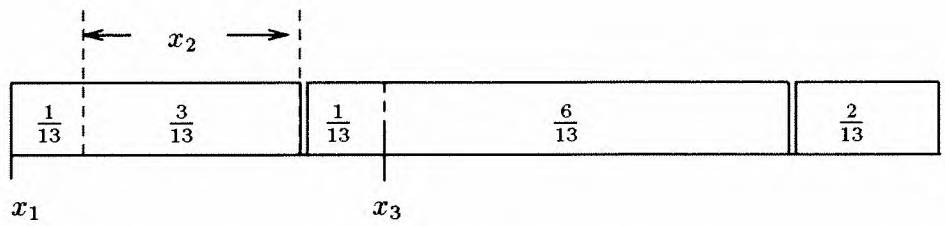


Figure 2.4: We show a line with $\mathbf{p} = (4/13, 7/13, 2/13)$ and constant velocity vector $\mathbf{v} = (1, 1, 2)$. If $x_3 = 5/13$ then we get the fixed allocation vector $(4/13, 4/13, 4/13)$ for any $x_2 \in [1/13, 4/13]$.

$t_{i+1}\left(x_{i+1}^p, \overline{x_{i+1}^p}\right)$ from expression 2.2, and a_i^p is an *inherited* allocation if $a_i^p = a_{i+1}^p$ from expression 2.3.

Note that all the time of a simple allocation is productive while a delay allocation includes time during which the worker is blocked and therefore idle. The last allocation a_n^p is always a simple allocation; and, since an item is completed at each reset, the time a_n^p between successive resets is the *cycle time* of the line and can differ at successive iterations. Further note that inherited allocations occur in "chains" of the form $a_{k-i}^p, \ldots, a_{k-1}^p, a_k^p$, with each allocation a_{k-i}^p inheriting the value of its successor a_{k-i+1}^p, until at the end of the chain there is a delay allocation a_k^p that determines the common value inherited along the chain.

For worker positions $\mathbf{x}^p$ we say that $w(\mathbf{x}^p) = \mathbf{a}^p$ is the corresponding (imputed) allocation of work, where the function w is given by expressions 2.1, 2.2 and 2.3. Now we can study the dynamics of a TSS line by both coördinate systems: In position space, given by its orbit $\{\mathbf{x}^0, \mathbf{x}^1, \ldots\}$, and in allocation space, given by its orbit $\{\mathbf{a}^0, \mathbf{a}^1, \ldots\}$.

An advantage of studying how the allocation of work changes is that allocation space is simpler in some respects than position space, yet it preserves important qualitative behavior. For example, if the TSS line is at a fixed point in one space

then it must be at a fixed point in the other. Note, however, that because of the possibility of blocking, the mapping w from $\mathbf{x}$ to $\mathbf{a}$ is, in general, many-to-one, so that to each point in allocation space there may correspond multiple points in the space of worker positions. For example, consider the line shown in Figure 2.4. Even though no allocations are inherited, with $x_3 = 5/13$, the same fixed allocation vector results for any $x_2 \in [1/13, 4/13]$.

The most immediately useful special structure in allocation space is as follows.

Theorem 2.2 *Any fixed point in allocation space has all entries equal.*

Proof Let $\mathbf{a}^0$ be a fixed point, and suppose that not all entries are equal. Let a_i^0 be the first non-inherited allocation that differs in value from a_n^0. If $a_i^0 < a_n^0$, then, by expression 2.2, $x_{i+1}^1 > x_{i+1}^0$; but then, since a_i^0 was not an inherited allocation, $a_i^1 > a_i^0$, contradicting the assumption that $\mathbf{a}^0$ was a fixed point. A similar argument handles the case in which $a_i^0 > a_n^0$. □

CHAPTER 3

TSS as an "accelerating" system

In our model a TSS system has complicated nonlinear dynamics that depend on the partition of work among the stations and the initial positions and velocities of the workers. Even the simplest imaginable nonlinear (in fact, even piecewise-linear) systems can exhibit extraordinarily complex and even chaotic behavior (Yoshida, Mori, and Shigematsu 1983). However, we show that, when TSS workers are sequenced from slowest to fastest, then the line becomes *self-organizing*: the workers will spontaneously and without intention space themselves so that the line produces at a constant rate and each worker repeatedly performs the same interval of work content.

Sequencing the workers from slowest to fastest is natural, because it helps avoid a slower worker blocking a faster one, with consequent loss of productive capacity. But our analysis shows a more profound benefit: that the line balances itself.

Define an *accelerating* TSS line to be one in which the workers are sequenced from slowest to fastest along the TSS line. More formally, we require that

$$\sup\left(\frac{v_i(x)}{v_{i+1}(x)}\right) < 1 \text{ for } i = 1, \ldots, n-1.$$

This condition is slightly stronger than the more immediately intuitive one that $v_i(x) < v_{i+1}(x)$ for all $x \in [0,1]$, so that successive workers are strictly faster at every component of work. We will need the stronger condition to avoid technical problems near any points at which $v_i(x)$ is discontinuous.[1]

We first show that an accelerating line has a unique fixed point in allocation space. We make use of the following technical lemma that shows that if **a** is a fixed point then one cannot form another fixed point of smaller cycle time unless worker velocities are increased. Such a lemma seems clear if all the allocations of **a** are simple, yet more care is needed if **a** has delay and inherited allocations.

Lemma 3.1 *Let* **a** *be a fixed point for workers* $v_1 < \cdots < v_n$ *on an accelerating TSS line. Let* **a'** *be any other allocation for a possibly different set of workers* $v'_1 < \cdots < v'_n$ *with* $v'_i \leq v_i$ *and* $a'_n < a_n$. *Then* **a'** *cannot be a fixed point.*

Proof First we observe that if $x_{i+1} < x'_{i+1}$ and if allocation a_i is simple, then $x_i < x'_i$: If this were not so, then we would have that $x_{i+1} - x_i < x'_{i+1} - x'_i$; but then since $v'_i \leq v_i$ and a_i simple, $a'_i > a_i$; and since **a** is a fixed point, $a_i = a_n > a'_n$, and so **a'** could not be a fixed point.

If all the allocations a_i are simple, then by the preceding paragraph, $x_1 < x'_1 = 0$, a contradiction. If some allocation of **a** is not simple, then the last one that is not simple must be a delay allocation a_j, with $j < n$. Since all of the succeeding allocations are simple, by the preceding argument it must be that $x_{j+1} < x'_{j+1}$. Furthermore, since a_j is a delay allocation, $x_j < x_{j+1}$. Let k be the index for which $x_j \leq P_{k-1} < x_{j+1} < P_k$; and let i be the index within **a'** for which $x'_i \leq P_{k-1} < x'_{i+1}$.

[1] Note that even when the workers are sequenced from slowest to fastest, items do not necessarily move more quickly as they near completion: it might be that all the workers are slower at the later stations than they are at the earlier stations.

Then $i \leq j$ because $x_{j+1} < x'_{j+1}$. If $i < j$, then a'_i is strictly greater than the time required for worker v'_{i+1} to travel from P_{k-1} to P_k, which, in turn, is strictly greater than the time required for worker v_j to travel from P_{k-1} to P_k, which is strictly greater than a_j; and if $i = j$, then $a'_i > a_j$ because $x_{j+1} < x'_{j+1}$ and $v'_i = v'_j \leq v_j$. In either case $a'_i > a_j = a_n > a'_n$, and again $\mathbf{a}'$ cannot be a fixed point. □

Theorem 3.2 *An accelerating TSS line has a unique fixed point in allocation space.*

Proof Suppose there were two distinct fixed points; then by Theorem 2.2, one must have strictly smaller entries, in contradiction to Lemma 3.1. □

In general the mapping from $\mathbf{x}$ to $\mathbf{a}$ is many-to-one as we saw from the example in Figure 2.4; however, the next theorem shows that this is not the case for an accelerating system.

Theorem 3.3 *For an accelerating TSS line, for the unique fixed point in allocation space there is a unique corresponding fixed point in position space.*

Proof Suppose, on the contrary, that $\mathbf{x}$ and $\mathbf{x}'$ are two distinct position vectors such that $w(\mathbf{x}) = w(\mathbf{x}')$ and let the $k+1$st element be the first element that differs between the two vectors. Without loss of generality assume $x_{k+1} < x'_{k+1}$.

If a_k is a simple or delay allocation then we get the contradiction that $a'_k > a_k$ since a'_k is strictly increasing with x'_{k+1}.

If however, a_k is an inherited allocation, then $a_k = a_z$ for some leading delay allocation a_z, $z > k$. Then $a_z = t_z\left(\underline{x_{z+1}}, x_{z+1}\right) + t_{z+1}\left(x_{z+1}, \overline{x_{z+1}}\right) < t_z\left(\underline{x_{z+1}}, \overline{x_{z+1}}\right)$. But then $a'_k > t_{k+1}\left(\underline{x_{k+1}}, \overline{x_{k+1}}\right) \geq t_z\left(\underline{x_{z+1}}, \overline{x_{z+1}}\right) > a_z$. Thus $\mathbf{x}$ and $\mathbf{x}'$ must be the same vector. □

3.1 Main convergence theorem

Our main result shows that an accelerating TSS line balances itself; that is, it reallocates work until each worker has an equal allocation that is independent of her starting position; and each worker repeats the same interval of work content.

Moreover, if there is no blocking, then all allocations are simple and so each worker invests the same clock time in each item produced.

Theorem 3.4 *If a TSS line is configured so that* $v_1 < \cdots < v_n$, *then the orbit of worker positions* $\{\mathbf{x}^p = f^p(\mathbf{x}^0)\}$ *converges to a unique fixed point.*

Proof

We will show that the orbit $\{\mathbf{a}^p\}_0^\infty$ converges to a unique allocation $\mathbf{a}^*$, all of whose entries are identical, and therefore by Theorem 3.3 converges to a unique fixed point in position space.

We define $r_{\max}(i) = \max_x \left\{\frac{v_i(x)}{v_{i+1}(x)}\right\}$. Note that since the TSS line is accelerating, $r_{\max}(i) < 1$.

We first prove a number of lemmata, the first shows that the maximum allocation in nonincreasing.

Lemma 3.5 *For each simple or delay allocation* a_i^{p+1}, *we have* $a_i^{p+1} \leq \max\{a_{i-1}^p, a_i^p, a_n^p\}$.

Proof At iteration p worker i starts at x_i^p and ends, just before the line resets, at x_{i+1}^{p+1}. Consider two cases for $i = 2, \ldots, n-1$.

1. a_i^{p+1} is a simple allocation.

3.1 Main convergence theorem

Our main result shows that an accelerating TSS line balances itself; that is, it reallocates work until each worker has an equal allocation that is independent of her starting position; and each worker repeats the same interval of work content.

Moreover, if there is no blocking, then all allocations are simple and so each worker invests the same clock time in each item produced.

Theorem 3.4 *If a TSS line is configured so that* $v_1 < \cdots < v_n$, *then the orbit of worker positions* $\{\mathbf{x}^p = f^p(\mathbf{x}^0)\}$ *converges to a unique fixed point.*

Proof

We will show that the orbit $\{\mathbf{a}^p\}_0^\infty$ converges to a unique allocation $\mathbf{a}^*$, all of whose entries are identical, and therefore by Theorem 3.3 converges to a unique fixed point in position space.

We define $r_{\max}(i) = \max_x \left\{\frac{v_i(x)}{v_{i+1}(x)}\right\}$. Note that since the TSS line is accelerating, $r_{\max}(i) < 1$.

We first prove a number of lemmata, the first shows that the maximum allocation in nonincreasing.

Lemma 3.5 *For each simple or delay allocation* a_i^{p+1}, *we have* $a_i^{p+1} \leq \max\{a_{i-1}^p, a_i^p, a_n^p\}$.

Proof At iteration p worker i starts at x_i^p and ends, just before the line resets, at x_{i+1}^{p+1}. Consider two cases for $i = 2, \ldots, n-1$.

1. a_i^{p+1} is a simple allocation.

(a) $x_i^{p+1} \geq x_i^p$

During iteration p worker i processes no less than $(x_{i+1}^{p+1} - x_i^{p+1})$ of the work content in time a_n^p. Since $a_i^{p+1} = t_i\left(x_i^{p+1}, x_{i+1}^{p+1}\right)$, then $a_i^{p+1} \leq a_n^p$.

(b) $x_i^{p+1} < x_i^p$

$a_i^{p+1} \leq a_n^p + t_i\left(x_i^{p+1}, x_i^p\right) < a_n^p + t_{i-1}\left(x_i^{p+1}, x_i^p\right) \leq a_{i-1}^p$.

2. a_i^{p+1} is a delay allocation.

Since a_i^{p+1} is a delay allocation, then $\underline{x_{i+1}^{p+1}} \leq x_{i+1}^{p+1} < \overline{x_{i+1}^{p+1}}$.

(a) $x_i^p \leq \underline{x_{i+1}^{p+1}}$

i. $x_{i+1}^{p+1} \geq x_{i+1}^p$

In time a_n^p worker $i+1$ processes no less than $\overline{x_{i+1}^{p+1}} - x_{i+1}^{p+1}$ prior to worker i processing $x_{i+1}^{p+1} - \underline{x_{i+1}^{p+1}}$. Since

$$a_i^{p+1} = t_i\left(\underline{x_{i+1}^{p+1}}, x_{i+1}^{p+1}\right) + t_{i+1}\left(x_{i+1}^{p+1}, \overline{x_{i+1}^{p+1}}\right),$$

we have $a_i^{p+1} \leq a_n^p$.

ii. $x_{i+1}^{p+1} < x_{i+1}^p$

If $x_i^p = \underline{x_{i+1}^{p+1}}$ and $x_{i+1}^p = \min\{x_{i+1}^p, \overline{x_{i+1}^{p+1}}\}$, we have

$$\begin{aligned} a_i^{p+1} &= a_n^p + t_{i+1}\left(x_{i+1}^{p+1}, x_{i+1}^p\right) \\ &= a_n^p + (a_i^p - a_n^p)\alpha, \end{aligned}$$

where α is the average velocity ratio of worker $i-1$ to worker i over the interval $[x_{i+1}^{p+1}, x_{i+1}^p]$. Thus we have

$$a_i^{p+1} \leq a_n^p + (a_i^p - a_n^p) r_{\max}(i)$$

or

$$a_i^{p+1} \leq r_{\max}(i)a_i^p + (1 - r_{\max}(i))a_n^p.$$

In addition, if $x_i^p < \underline{x_{i+1}^{p+1}}$ then we may have underestimated at least one of a_i^p and a_n^p, and if $x_{i+1}^p > \overline{x_{i+1}^{p+1}}$ then we have underestimated a_i^p by our assumption. Thus our inequality still holds.

(b) $x_i^p > \underline{x_{i+1}^{p+1}}$

Worker i begins and ends iteration p at the same station. Thus

$$a_{i-1}^p \geq t_{i-1}\left(\underline{x_i^p}, x_i^p\right) + t_i\left(x_i^p, \overline{x_i^p}\right)$$

and

$$\begin{aligned} a_i^{p+1} &= t_i\left(\underline{x_{i+1}^{p+1}}, x_{i+1}^{p+1}\right) + t_{i+1}\left(x_{i+1}^{p+1}, \overline{x_{i+1}^{p+1}}\right) \\ &= t_i\left(\underline{x_i^p}, x_{i+1}^{p+1}\right) + t_{i+1}\left(x_{i+1}^{p+1}, \overline{x_i^p}\right) \\ &< a_{i-1}^p, \end{aligned}$$

since $v_{i-1} < v_i < v_{i+1}$.

$a_n^{p+1} \leq a_n^p$ or $a_n^{p+1} < a_{n-1}^p$ follows as in Case 1. $a_1^{p+1} \leq a_n^p$ if a_1^{p+1} is a simple allocation and $a_1^{p+1} < a_1^p$ or $a_1^{p+1} \leq a_n^p$ follows as in Case 2 if a_1^{p+1} is a delay allocation.

□

The following Lemma gives a tighter bound on changes in the allocation of worker n at phase $p+1$ when worker $n-1$ does not reach x_n^p during phase p.

Lemma 3.6 *If* $x_n^{p+1} < x_n^p$ *then*

$$a_n^{p+1} = \alpha a_{n-1}^p + (1 - \alpha)a_n^p,$$

where $\alpha < r_{\max}(n-1)$.

Proof If a_{n-1}^p is a delay allocation then there is an interval $[s_1, s_2]$ with $x_{n-1}^p \in [s_1, s_2]$ for which a_{n-1}^p retains its value. Since worker $n-1$ must reach s_2 during

phase p, we can assume $x_{n-1}^p = s_1$ without altering a_n^{p+1}, and thus worker $n-1$ is not delayed during phase p and the following calculation is simplified:

$$\begin{aligned} a_n^{p+1} &= a_n^p + t_n\left(x_n^{p+1}, x_n^p\right) \\ &= a_n^p + (a_{n-1}^p - a_n^p)\alpha, \end{aligned}$$

where α is the average velocity ratio of worker $n-1$ to worker n over the interval $[x_n^{p+1}, x_n^p]$. □

Since $a_i^p \leq \max\{a_{i-1}^{p-1}, a_i^{p-1}, a_n^{p-1}\}$ from Lemma 3.5, we can construct at least one substochastic matrix $\mathbf{T}_p$ such that $\mathbf{a}^p = \mathbf{T}_p\mathbf{a}^{p-1}$. We append a dummy column to each $\mathbf{T}_p$ to make up any row deficiencies and add a dummy row $[0, \ldots, 0, 1]$ so that each augmented $\mathbf{T}_p$ is a stochastic matrix. We must also add a corresponding dummy allocation a_0 which retains value 0 at each iteration.

We can now use a Markov Chain metaphor to analyze the dynamics of the system by interpreting each $\mathbf{T}_p$ as a matrix of one-step transition probabilities. To analyze the allocation vector of iteration p we have $\mathbf{a}^p = \mathbf{T}_p\mathbf{T}_{p-1}\cdots\mathbf{T}_1\mathbf{a}^0$. The Markov Chain, however, actually makes transitions in an order opposite from TSS iterations, and thus the Markov Chain makes its first transition with matrix $\mathbf{T}_p$, the second with $\mathbf{T}_{p-1}$, and so forth to its last transition according to the matrix $\mathbf{T}_1$. This mild dissonance in notation is harmless, however, because we are interested only in the product $\mathbf{T}_p\mathbf{T}_{p-1}\cdots\mathbf{T}_1$ which is independent of how the indices are interpreted.

In keeping with Markov Chain terminology we will refer to the Chain as having $n+1$ "states", and thus one should not confuse this terminology with the "state" of a dynamical system (which, for our system, we describe by a position or allocation vector). For convenience we will refer to a random process at "iteration" p instead of the more conventional terminology, "time" p, for Markov Chains. Thus at every

iteration, corresponding to the allocation of each worker, is a state of the Markov Chain, and so we refer to a state at a particular iteration as being simple, delay or inherited.

We next define rules of how to form each transition matrix $\mathbf{T}_p$. In short, we show that each row $i < n$ must have a positive element in at least one of column $i-1$, i, $i+1$, or n. Furthermore, we define a constant $r < 1$ such that any element (i,i) is either less than r or can exceed r only a finite number of times in succession. In addition any element $(i,i+1)$ is less than r. For row n we always have an element greater than a positive constant in column n or the dummy column 0. We are able to show, through a coupling argument, that any two random processes must meet in finite time at state n if they do not transit to the absorbing state 0. The theorem then follows once we show that the allocation for the last worker must converge to a positive constant.

The rules for forming each $\mathbf{T}_p$ are as follows. We use Lemma 3.5 to write three types of transitions from a simple or delay state $i < n$. The first type of transition is from state i to the absorbing state 0 with some probability $\lambda \geq 0$ and to state $i-1$ with probability $1-\lambda$. A second type of transition is from i to state 0 with some probability $\lambda \geq 0$ and to state n with probability $1-\lambda$. Finally by Case 2(a)ii of Lemma 3.5, the system can transit to state 0 with probability $1-\lambda_1 r_{\max}(i)-\lambda_2(1-r_{\max}(i))$, from state i back to state i with probability $\lambda_1 r_{\max}(i)$, and to state n with probability $\lambda_2(1-r_{\max}(i))$, where $0 < \lambda_1 \leq 1$, $0 < \lambda_2 \leq 1$. If state i is inherited we simply mimic the transitions of state $i+1$. For instance, if delay allocation k has a transition to states n and 0, all of the inherited allocations immediately preceding k will also have transitions of the same probability to states n and 0.

We handle transitions from state n by first selecting a small $0 < \epsilon < r_{\max}(n-1)$.

If $x_n^{p+1} \geq x_n^p$ then we transition from n to the absorbing state 0 with probability $\lambda \geq 0$ and back to n with probability $1 - \lambda$. If $x_n^{p+1} < x_n^p$ then we follow Case 1b of Lemma 3.5 and transition from n to the absorbing state 0 with probability $\lambda > 0$ and to $n - 1$ with probability $1 - \lambda$ only if $\lambda \geq \epsilon$. Otherwise we utilize Lemma 3.6 and transit to state $n - 1$ with probability no greater than $r_{\max}(n - 1)$ and back to n with probability at least $1 - r_{\max}(n - 1)$. In this way we have, at each iteration, either a self-loop transition from state n of probability at least $1 - r_{\max}(n - 1)$ or a transition from state n to state 0 of probability at least ϵ.

We now make a technical observation.

Observation 3.1 *Case 2b of Lemma 3.5 can occur at most m times in succession for any allocation.*

Observation 3.1 follows since Case 2b of Lemma 3.5 requires a worker to begin and end an iteration *on* the same work station and therefore he must start the next iteration prior to the beginning of that work station.

The next lemma shows that there is a finite number u such that the product of any u consecutive transition matrices must have, for all rows, an element in column n or 0 greater than a positive constant.

Lemma 3.7 *Consider any random process in state i at iteration p. The probability of transition to state n or absorption to state 0 within the next u transitions, where $u = (m + 1)(n - 1)$, is greater than some constant $q > 0$.*

Proof We consider the random process starting in any state $i < n$. To avoid eventual absorption by 0 or transition to n within $n - 1$ steps, the process must make a self-loop transition from some state h or a forward transition from some

state h to a state of larger index. Self-loop transitions must be from either a delay state or an inherited state and a forward transition can occur only from an inherited state.

An inherited allocation h will have a self-loop transition only if the leading delay allocation transits to his predecessor as suggested by Case 2b of Lemma 3.5. But from Observation 3.1 only m such transitions can occur consecutively from any state h, at which point a self-loop from a delay state, a forward transition, a transition to 0, to $h-1$, or to n must occur.

From Case 2(a)ii of Lemma 3.5, a self-loop transition from a delay state h has probability no greater than $r_{\max}(h)$. A forward transition occurs only from an inherited state when the leading delay state has a self-loop, and thus such forward transitions have probability no greater than $r = \max_i\{r_{\max}(i)\}$. Thus any process that avoids delay self-loops and forward transitions for $u = (m+1)(n-1)$ steps must either have been absorbed by 0 or transited to n. Thus our lemma follows with $q = (1-r)^u$. □

The next lemma establishes that the cycle time of the line converges to a constant.

Lemma 3.8 *The sequence $\{a_n^p\}_{p=0}^{\infty}$ converges to some positive constant.*

Proof Suppose on the contrary that the allocation for worker n does not converge. Then we must incur an infinite number of transitions from state n to state 0 of probability greater than any small $\epsilon < r_{\max}(n-1)$, and we use this ϵ in our rules for constructing the transitions from state n. From Lemma 3.7 there is a probability of at least q that any process will either absorb or transit to n within u transitions

prior to each such ϵ transition. Therefore by Lemma 3.6 we have a probability of at least $\epsilon q(1-r_{\max}(n-1))^u$ of the process being absorbed by state 0 each time we have an absorption at least ϵ from n to 0. Therefore an infinite number of such transitions from state n to state 0 of probability greater than ϵ implies, by a geometric argument, that all processes must eventually absorb, and thus $\mathbf{T}_p\mathbf{T}_{p-1}\cdots\mathbf{T}_1$ must converge to the zero matrix (with a column of ones for the absorbing state). But this is impossible because $\sum a_i \geq t_n(0,1) > 0$. Thus the absorption probabilities for state n must tend to zero and therefore the allocation for worker n converges to some constant. This constant must be positive since the cycle time cannot tend to zero. □

We can interpret row n of the sequence $\mathcal{T}^{(p)} = \mathbf{T}_p\mathbf{T}_{p-1}\cdots\mathbf{T}_1$ as the vector of probabilities that after p iterations the process is in any state j after starting from state n. Let $\mathcal{T}_{i0}^{(p)}$ be the component in the ith row corresponding to the column for the absorbing state 0. We let $\mathcal{T}_i^{(p)}$ be ith row vector and $\mathcal{T}_{i/0}^{(p)}$ be the ith row vector excluding the component $\mathcal{T}_{i0}^{(p)}$.

Lemma 3.9 *The sequence* $\{a_i^p\}_{p=0}^{\infty}$ *converges to some constant for each* $i = 1,\ldots,n$.

Proof The component $\mathcal{T}_{i0}^{(p)}$ converges for each i since its value is nondecreasing and bounded above by 1. Then $f_{i0} = \lim_{p\to\infty}\mathcal{T}_{i0}^{(p)}$ is interpreted as the probability a process starting in state i is eventually absorbed by state 0. Since, from Lemma 3.8, $\mathcal{T}_n^{(p)}\mathbf{a}^0$ converges to a positive constant we know $f_{n0} < 1$. So we can consider a process z_n that starts in state n, but is not eventually absorbed by state 0. The behavior of such a process determines the values in the vector $\mathcal{T}_{n/0}^{(p)}$.

We consider processes starting in any state $i < n$. If $f_{i0} = 1$ then $\mathcal{T}_i^{(p)}$ converges

to $[0, \ldots, 0, 1]$, and thus $\mathcal{T}_i^{(p)}\mathbf{a}^0$ converges to zero. If $f_{i0} < 1$ then let z_i be a process which starts in state i and is not eventually absorbed by state 0. We show through a coupling argument that z_n and z_i must meet in finite time and thus $\mathcal{T}_{i/0}^{(p)}$ must converge to a constant multiple of $\mathcal{T}_{n/0}^{(p)}$. By Lemma 3.6 and Lemma 3.7 process z_i will meet z_n at state n with probability at least $q(1 - r_{\max}(n-1))^u$ before z_n leaves state n. If they fail to meet in the first u transitions, then by Lemma 3.7, one of processes will again transit to state n in finite time and therefore, by a geometric argument, z_i and z_n must eventually meet. Therefore $\mathcal{T}_{i/0}^{(p)}$ converges to $((1 - f_{i0})/(1 - f_{n0}))\mathcal{T}_{n/0}^{(p)}$, and thus $\mathcal{T}_i^{(p)}\mathbf{a}^0$ converges to a constant as p grows large. □

From Lemma 3.9 we have for any starting allocation $\mathbf{a}^0$ that the sequence of work allocations converges to $\mathbf{a}^* = \mathcal{T}^{(p)}\mathbf{a}^0$ as p grows large. But we know from Theorem 3.2 that $\mathbf{a}^*$ must be the unique fixed point for the system.

□

3.2 The dynamics of an accelerating line

Figures 3.1, 3.2 and 3.3 show the convergence of a system from three complementary points of view. Figure 3.1 shows an example of how the movement of the workers stabilizes, with the faster workers allocated more work; Figure 3.2 shows the convergence of the system within the state space of worker positions; and Figure 3.3 shows the average production rate converging. These simulations were generated by three workers of constant velocity $\mathbf{v} = (1, 2, 3)$.

The next theorem gives some insight into how an accelerating TSS line converges.

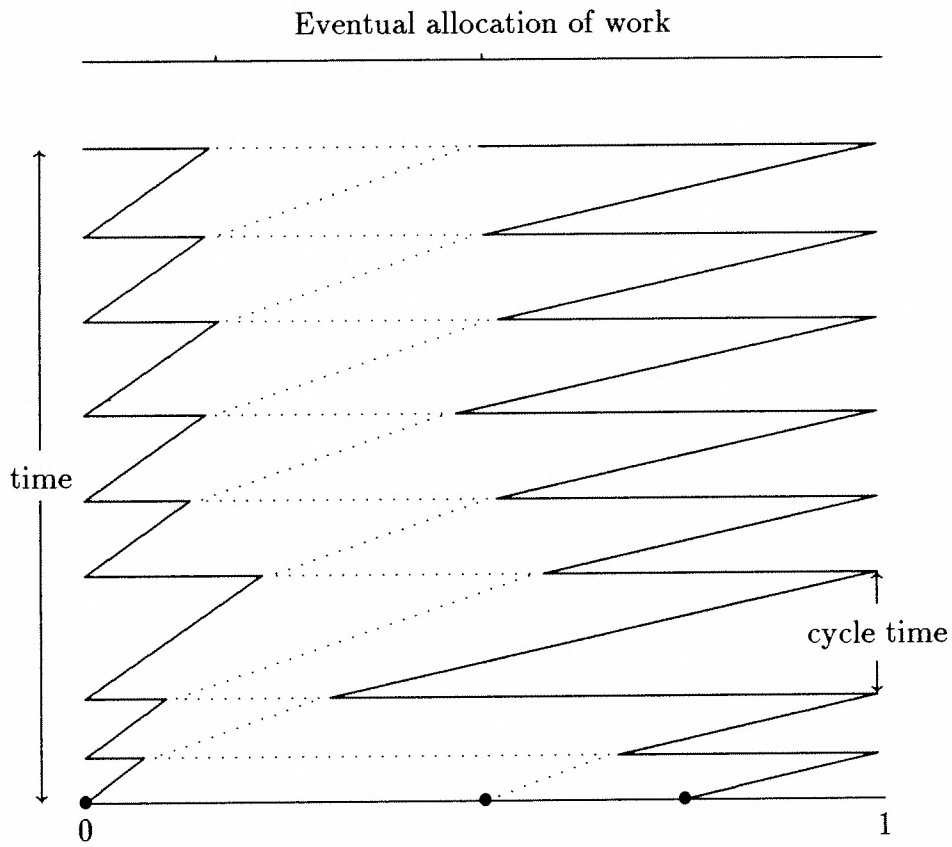


Figure 3.1: A time-expanded view of a TSS production line with three workers sequenced from slowest to fastest. The solid horizontal line represents the total work content of the product and the solid circles represent the initial positions of the workers. The zigzag vertical lines show how these positions change over time and the rightmost spikes correspond to completed items. In this simulation the system quickly stabilized so that each worker repeatedly executes the same portion of work content of the product.

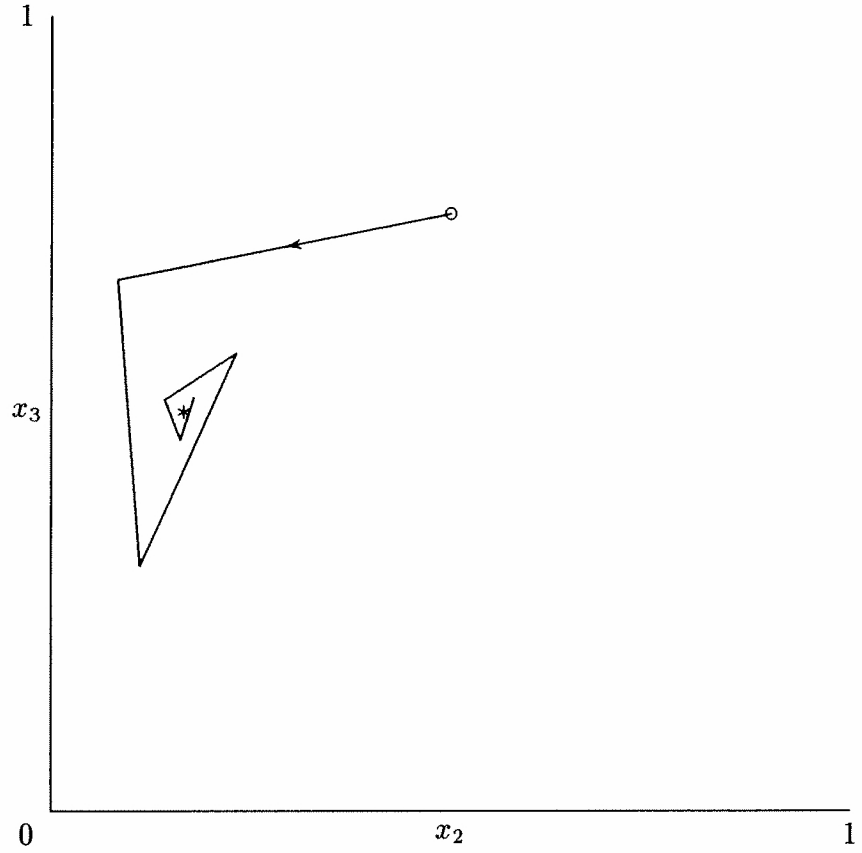


Figure 3.2: The positions of the workers on the production line at successive instants when it resets. (Since the position of the first worker is always 0 when the line resets, only (x_2, x_3), the positions of the second and third workers, are plotted here.) From any initial position the system converges. Here the fixed point is $(1/6, 1/2)$.

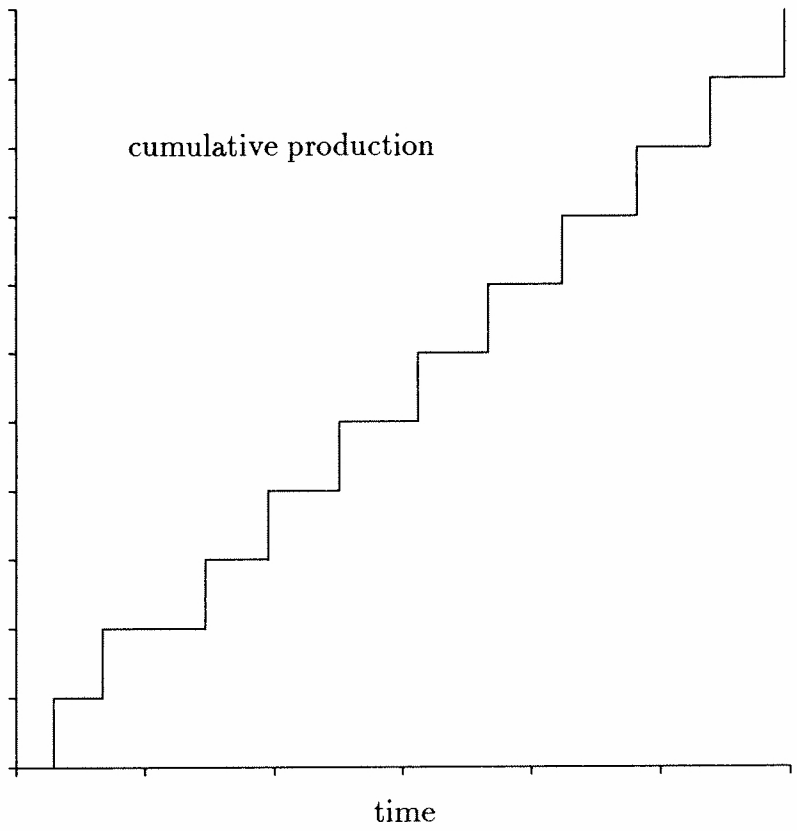


Figure 3.3: When workers are sequenced from slowest to fastest, the cycle time converges to a unique value, independently of where the workers start on the line. In this instance, as the system approaches its limiting configuration, no worker is ever blocked, and the production rate approaches 6, the maximum possible.

Theorem 3.10 *The maximum allocation $a^p_{\max} = \max_i\{a^p_i\}$ is non-increasing in p and will strictly decrease at least every n iterations unless it reaches the common allocation value at the fixed point.*

Proof From Lemma 3.5 the maximum allocation is non-increasing. From the cases within the proof of Lemma 3.5, if $a^p_n < a^p_{\max}$ then $a^{p+1}_{\max} < a^p_{\max}$. Otherwise let j index the last allocation such that $a^p_j < a^p_{\max}$. Then from Lemma 3.5, if a^{p+1}_{j+1} is a simple allocation then $a^{p+1}_{j+1} < a^p_{\max}$. In this way, any allocation $a^p_j < a^p_{\max}$ will propagate an allocation to the last worker such that $a^{p+k}_n < a^p_{\max}$, $k < n$, as long as this chain is built of simple allocations. Once $a^{p+k}_n < a^p_{\max}$ then $a^{p+k+1}_{\max} < a^p_{\max}$. Therefore, the only way to prevent the maximum allocation from decreasing within n iterations is for a chain of allocations, $a^p_z, a^p_{z+1}, \ldots, a^p_n$ all equal to $a^p_{\max}$ to form with a^p_z being a delay allocation. But if such a chain occurs it will persist forever. This is so since workers $z+1, \ldots, n$ will return to the same position at the next phase. And since the allocation of worker $z-1$ will never exceed $a^p_{\max}$, worker z must always return with the same delay allocation of $a^p_{\max}$. Thus the maximum allocation will decrease at least every n iterations or reach the common value at the fixed point. □

Consider viewing each allocation $\mathbf{a}$ as a static allocation of work as in a classical assembly line. That is, each worker i produces exactly the interval $[x_i, x_{i+1}]$ and thus an item is produced every $a_{\max}$ time units. Then Theorem 3.10 shows that the TSS line makes steady progress from one classical assembly line partition of work to another. The fixed point is then the allocation where each worker i takes the same time to produce her interval $[x_i, x_{i+1}]$.

Theorem 3.4 suggests that an accelerating TSS line is robust in several senses. First, it will balance itself independently of the starting positions of the workers. It

will also *re*-balance itself after a one-time disruption; for example, when a worker takes a break, the work content will be spontaneously reallocated among the remaining workers. It will also continually rebalance itself in the presence of noise, such as small variance in the time to complete a task. Furthermore, the line will rebalance itself to account for "drift", such as when workers tire and slow (as long as the workers remain sequenced from slowest to fastest). Finally, the line is self-balancing without knowing the statistics of task times or even worker velocities; all that is required is to know the *relative* velocities of the workers (who is faster).

It is important to note that for actual production lines we can relax the requirements for a line to be accelerating by not requiring every worker $i+1$ to be faster than i over all points of the production line. For instance it clearly does not matter the velocity of the first worker on the last station — she will never operate it. In fact, worker $i+1$ only really needs to be faster than i over the station or two that they typically share. Then under reasonable noise, if workers are begun near the fixed point then the line will behave just as if it were a truly accelerating line.

3.3 Production rate

An accelerating line will not necessarily produce as large a production rate as is possible with a different ordering of workers. However, the following theorem bounds the performance of an accelerating line as compared to any other sequence of workers.

Theorem 3.11 *The fixed point production rate of an accelerating line is always within a factor n of the best achievable by any other sequence of workers.*

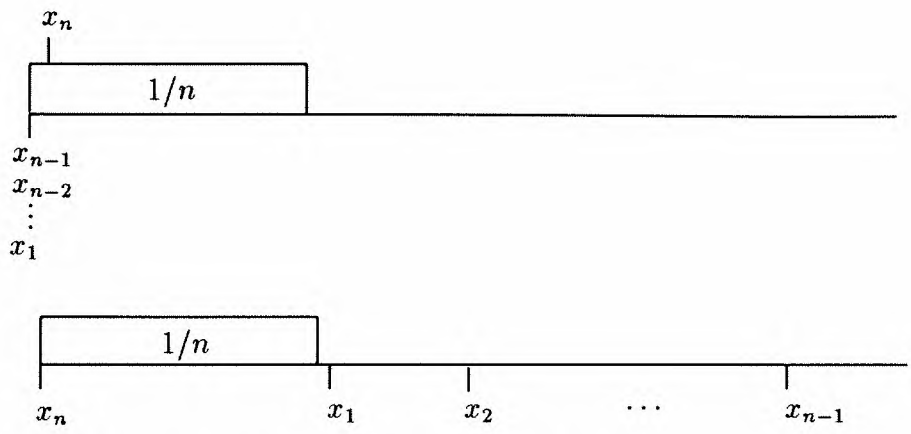


Figure 3.4: The first accelerating line is a factor of n worse than the second non-accelerating line. Workers $1, \ldots, n-1$ are queued for station 1 in the first line. The fastest worker n has a constant velocity of 1. Each other worker $i < n$ has constant velocity $i\epsilon$ on the first station and constant velocity $1-(n-i)\epsilon$ on stations $2, \ldots, m$. The first station has processing requirement $1/n$, the remaining stations have small processing times.

Proof Rescale each Δx of time so that $v_n(x) = 1$ for all x and thus each function $v_i(x)$, $i < n$, must also be adjusted accordingly. For $i < n$ we now have $v_i(x) < 1$ for all x. The fixed point for the accelerating system always keeps worker n busy and must therefore achieve a production rate of at least 1. Since each $v_i < 1$, any other sequence of workers has a production rate no more than n. □

Figure 3.4 shows an example where this bound is tight. The fixed point of the accelerating line almost exclusively uses worker n since the velocity of the first workers is so poor on the first station. The second line is able to use the first workers beyond station 1 and therefore substantially improve the production rate. The production rate of the accelerating line tends to 1 as ϵ tends to 0. The production rate of the second line at a fixed point keeps all workers busy, with workers $1, \ldots, n-1$ beyond station 1, and thus tends to n as ϵ goes to 0. This fixed point for the second line is

realized with each worker i processing an interval of width

$$\frac{1-(n-i)\epsilon}{n\left(1-\frac{(n-1)\epsilon}{2}\right)}.$$

The following example shows that a sequence other than slowest-to-fastest can be arbitrarily less productive than the slowest-to-fastest sequence of workers. Consider a TSS line with $\mathbf{p} = (\epsilon, 1-\epsilon)$ and two workers, one of constant velocity ϵ and the other of constant velocity $1-\epsilon$. In this sequence the workers achieve a production rate of one item per time unit; but reversing the sequence gives a production rate of one item per $(1-\epsilon)/\epsilon$ time units, which can be made arbitrarily small. Thus the worst-case ratio of production rates is unbounded above.

Determining the fixed point production rate of an accelerating line is not trivial. In fact, the easiest method seems to be an iterative method that starts with any position vector $\mathbf{x}^0$ and computes each successive position vector $\mathbf{x}^{p+1} = f(\mathbf{x}^p)$ by simulating the TSS rule. The procedure is terminated with production rate $1/a_n^p$ when $|\mathbf{x}^{p+1} - \mathbf{x}^p|$ is sufficiently small. Unless the velocity functions are unduly complex, such a procedure is easily coded in any high-level language.

3.4 Ordering of work stations

The ordering of the stations can affect the production rate of an accelerating line. Figure 3.5 shows an improvement in production rate when stations are ordered by increasing processing times. One is now tempted to conjecture that the production rate of an accelerating line might always be at its maximum when stations are ordered by nonincreasing processing times. However, a simple ordering of the work stations in this manner, that is independent of the velocity functions, is, in general,

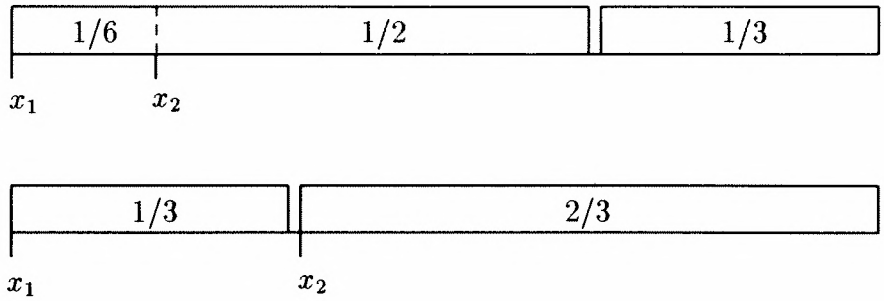


Figure 3.5: The ordering of the stations can affect the production rate. We show a two station line with two workers of constant velocity vector $\mathbf{v} = (1, 2)$. In the first line the stations are ordered $\mathbf{p} = (2/3, 1/3)$ and the resulting fixed point production rate is 12/5. In the second line the stations are ordered $\mathbf{p} = (1/3, 2/3)$ and both workers remain busy at the fixed point and thus achieve an improved production rate of 3.

meaningless. This is because one can simply increase(decrease) a given p_j and make a corresponding decrease(increase) in the velocity functions over the interval for station j — such a change still maintains an accelerated ordering of the workers.

CHAPTER 4

Complicated behavior

We have determined by computational experiment that, when workers are sequenced other than from slowest to fastest, (our model of) a TSS line will in general fail to balance itself. Instead, its qualitative behavior can depend on the initial positions of the workers. Typically the line becomes trapped in periodic behavior with a suboptimal production rate. There is, as we showed in Theorem 2.1, always a stable allocation of work; the trouble is that the line can fail to converge to it. In fact, the fixed point can be a *repeller*, so that if the system ever deviates, however slightly, from it, then the system must inexorably diverge from it (Devaney, 1989). In the real world, this deviation *must* occur because the data that determine the fixed point—for example, worker velocities—are not knowable exactly, and, moreover, might change over time.

Figures 4.1 and 4.2 show a simulation, generated by workers of constant velocity $\mathbf{v} = (3, 1, 2)$, in which the fixed point is a repeller and any orbit that strays must eventually be trapped by a limit cycle with production rate less than that of the fixed point. The suboptimality results from the fact that a faster worker can be repeatedly blocked by a slower worker, with consequent waste of productive capacity. In other

simulations we have found instances of quite large cycles, some at the limits of the numerical resolution of our computer and of the patience of the observers. For example, for constant velocities $\mathbf{v} = (2, 1, 100/51)$ there is an attracting limit cycle of length 1159.

Furthermore, even slight changes in the data (distribution of work content over the stations, initial positions of the workers, and, especially, values of the v_i) could result in wildly varying behavior of the line.

Figure 4.3 shows an example of a line with multiple fixed points in allocation space with differing production rates (for a fixed set and sequence of workers). Such a phenomena can occur because an allocation a_i can decrease as x_{i+1} increases; in this example a_3 decreases as x_4 increases. (Note that for an accelerating line the allocation for worker i is always strictly increasing in x_{i+1}.)

In practice, repeated, complicated reallocation of work is not necessarily a problem as long as each worker continues to visit the same subset of stations. Then it is possible that the only effect is, for example, that a worker might preëmpt the sewing of a long seam at different spots in successive resets. However, if the work allocations vary so that workers visit a changing set of stations, then the line can be difficult to manage. For example, in simulations some workers visited most of the stations while other colleagues were blocked by slower workers and visited few stations.

To enforce manageability, commercial TSS lines are set up with a meta-rule that restricts each worker to a predetermined contiguous subset of stations. This has some (at least theoretical) disadvantages: It can waste productivity by forcing a worker to be idle; it reduces the flexibility of the line by making it more involved to add or remove workers; and it might not be necessary if workers are sequenced

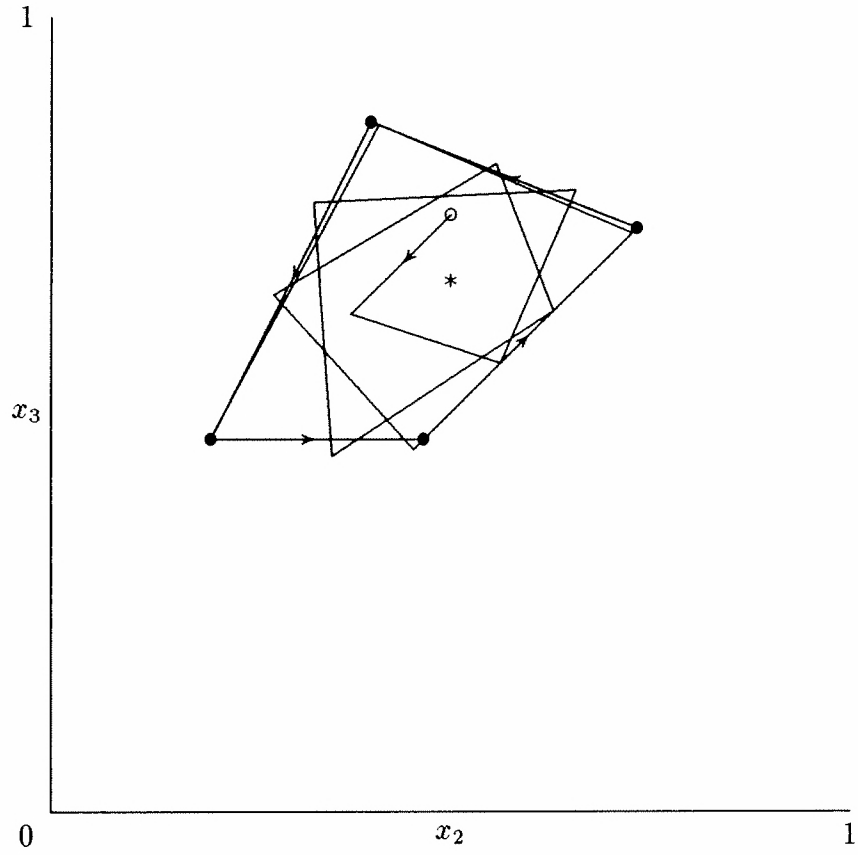


Figure 4.1: The positions of the workers on the production line at successive instants when it resets. In this instance the fixed point $(1/2, 2/3)$, indicated by $*$, achieves the largest possible rate of production, but is a repeller, and any orbit that strays from it will be trapped by the attracting but suboptimal limit cycle consisting of the points $(3/15, 7/15)$, $(7/15, 7/15)$, $(11/15, 11/15)$, and $(2/5, 13/15)$, indicated by $\bullet$'s.

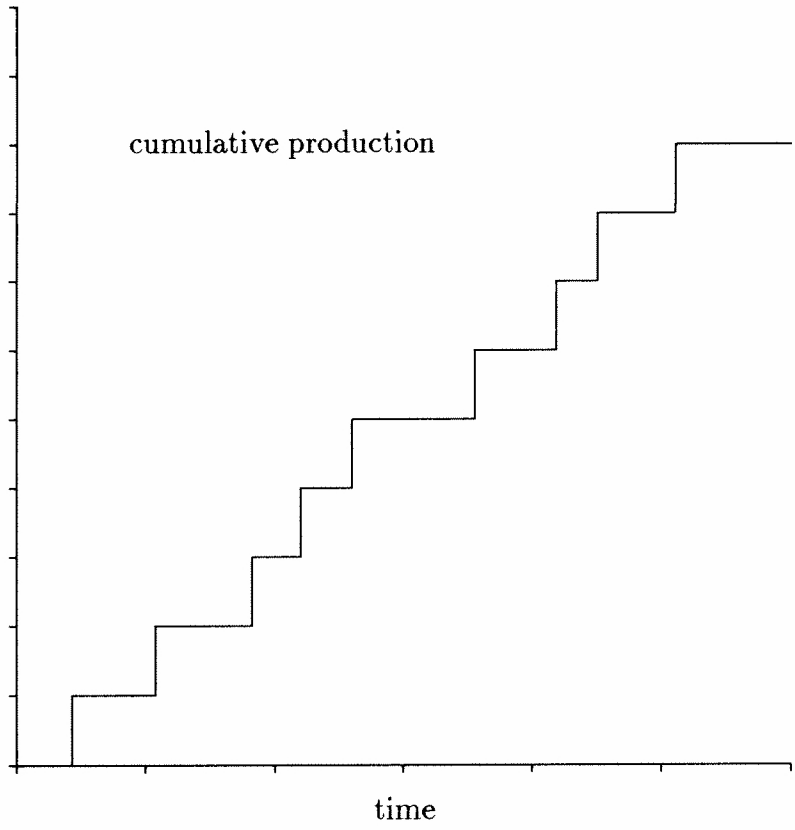


Figure 4.2: As the system is trapped by a limit cycle, the cycle time oscillates and the average production rate converges to 60/11, which is less than the optimum value of 6.

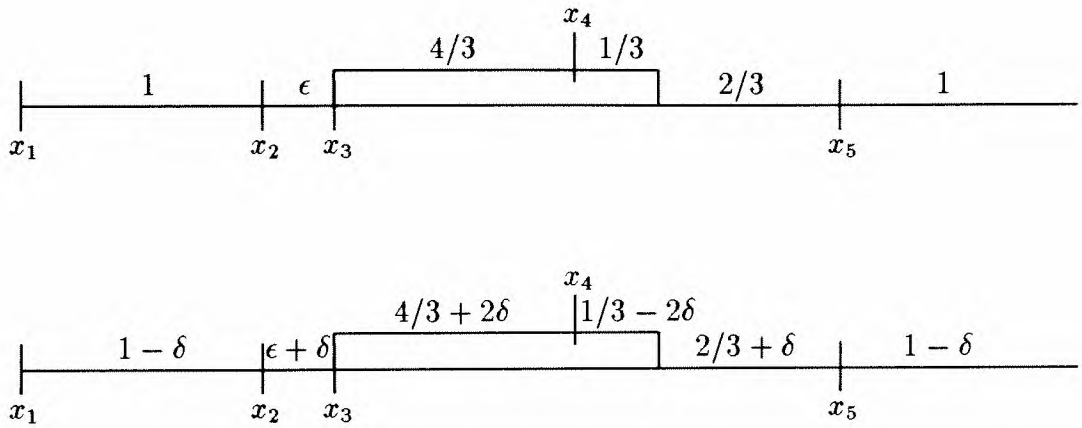


Figure 4.3: We show multiple fixed points in allocation space by considering this five worker line with constant velocity vector $(1, 1, 2, 1, 1)$. All stations have small processing requirements except the one depicted with a processing requirement of $5/3$. The first line has position vector $(0, 1, 1 + \epsilon, 7/3 + \epsilon, 10/3 + \epsilon)$ for some small ϵ. All allocations are equal to 1 with the allocation of the second worker being inherited and the third worker being delayed. The next line shifts the positions of each worker by a small δ or 2δ to get a fixed point with common allocation value equal to $1 - \delta$.

from slowest to fastest.

More troubling than complicated behavior is anomalous behavior: If the workers are not sequenced from slowest to fastest, then adding a worker to the line can *decrease* the production rate! For example consider the lines in Figure 4.4; by adding a worker the production rate drops from 4 to $8/3$. Such an example shows that the addition of a slow worker caused the first worker to become idle, and thus such an addition, in general, can worsen the production rate by an arbitrary amount as the velocity of the idled worker is made arbitrarily large. Similarly, the production rate of the line $\mathbf{p} = (1/2, 1/4, 1/4)$ with workers of constant velocity $\mathbf{v} = (2, 1, 1)$ decreases from 4 to $8/3$ if worker 3 doubles her velocity. The production rate decreases because, after the ostensible improvements, the first worker is always blocked, while the second worker, who is the slowest, is blocked during part of each

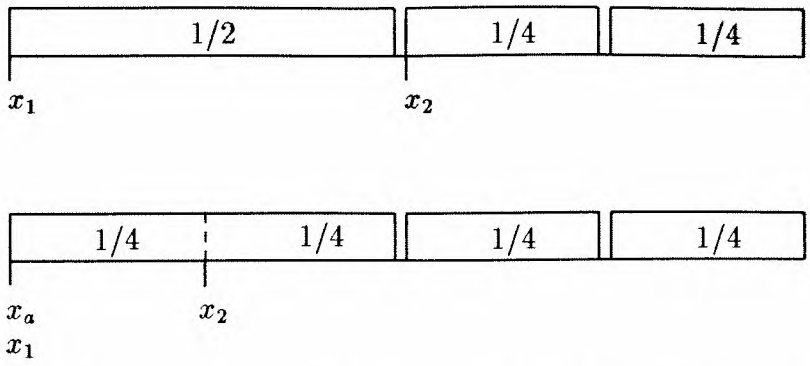


Figure 4.4: The addition of a worker decreases the production rate for this line with processing times $\mathbf{p} = (1/2, 1/4, 1/4)$. The first line has two workers of constant velocity $\mathbf{v} = (2, 2)$ that achieves a production rate of 4. The production rate of the second line decreases to 8/3 when an additional worker, a, of velocity 1 is inserted between the two original workers.

iteration.

We emphasize that system behavior is neither complicated nor anomalous when workers are sequenced from slowest to fastest.

Theorem 4.1 *If workers on a TSS line are maintained in sequence from slowest to fastest, then adding or speeding up a worker will never decrease the production rate.*

Proof At the expense of complicated notation, Lemma 3.1 can be extended to hold whenever there are no more than n of the slower workers. Then this result holds as an immediate corollary. □

Incidentally, we have been told that commercial TSS lines are frequently configured with the fastest workers at the very first and very last positions. If this actually increases the production rate, we conjecture that it is due to psychological factors. Our model predicts no benefits from putting a fast worker first on the line.

Determining the production rate of a TSS line seems very complicated. For one, it can be dependent on the starting positions of the workers. The line may converge

to one of multiple fixed points or converge to a limit cycle of extraordinary length. Chaotic behavior may also be possible, although it is not clear if we have experienced such behavior from our experiments.

CHAPTER 5

Special cases of the TSS line

5.1 A uniform accelerating system

In Chapter 3 we studied an accelerating system for general velocity functions $v_i(x)$. While we could show such systems converged to a unique fixed point, we were unable to gain much insight (short of simulating the line) into a number of important questions: What is the production rate of the line? How many workers should be used? Will reordering the stations improve the production rate? We explore a simpler velocity function, that is still powerful enough to approximate an actual production line, that will enable us to develop simple expressions for the fixed point in allocation and position space. This will allow a manager, whose line is close to fitting such a model, to plan the number of workers, to plan the training each worker will require, and predict the production rate of the line. This model will also provide insight into how work stations should be ordered.

In this model each worker i has a velocity that is constant or uniform over the entire unit of work, so that $v_i(x) = v_i$, and we maintain an accelerating system so that $v_1 < v_2 < \cdots < v_n$. Therefore, in this model, each worker can be viewed as

moving smoothly down the line at a constant rate until she is blocked or the line resets. While an accelerating model is a good model for the apparel industry since the workers are skilled and the tasks are similar, the drawback of the uniform accelerating model is clearly that it assumes such differences in skill translate uniformly from one task to another. However, managers of apparel lines that we spoke with agreed that this model is close enough to be useful.

We again normalize the length of the line to 1 and assume preëmption costs to be zero.

Lemma 5.1 *The largest possible production rate for a line with workers of uniformly different speeds is* $\sum_{i=1}^{n} v_i$.

Proof Since worker i can produce at a rate no faster than v_i items per time unit, the lemma follows by summing this maximum production rate for each worker. □

5.1.1 a non-blocking model

None of the apparel lines that we saw had workers being blocked on a regular basis. Such blocking is usually a sign of an overstaffed line or possibly an indication that the stations are not configured properly. We will therefore study fixed points for the uniform accelerating system that are most realistic for actual production lines — ones for which no worker is blocked.

Since we are concerned with the structure of the non-blocking fixed point, we can further simplify our analysis by assuming no workers are ever blocked; that is, there are no restrictions on sharing a work station. This model also describes having n copies of each work station in parallel.

We already know from Theorem 3.4 that this system will converge to a fixed point. However, the proof of convergence for this system is both simple and instructive and will provide additional insight that is hidden in the convergence proof of the general model. Furthermore, the proof of convergence is a means to derive simple expressions for the fixed allocation and position vector.

For this model the allocation of worker i is always simple and therefore

$$a_i^p = (x_{i+1}^p - x_i^p)/v_i.$$

This leads to the following lemma describing the allocations from one iteration to the next.

Lemma 5.2 *For the uniform model of the TSS line we have*

$$a_1^{p+1} = a_n^p$$

and

$$a_i^{p+1} = \left(\frac{v_{i-1}}{v_i}\right) a_{i-1}^p + \left(1 - \frac{v_{i-1}}{v_i}\right) a_n^p \quad \textit{for } i = 2, \ldots, n.$$

Proof The allocation for worker 1 holds since she is never blocked. For $i > 1$ we have

$$\begin{aligned} a_i^{p+1} &= \frac{(x_i^p + v_i a_n^p) - (x_{i-1} + v_{i-1} a_n^p)}{v_i} \\ &= (x_i^p - x_{i-1}^p)/v_i + (1 - v_{i-1}/v_i)\, a_n^p \\ &= \left(\frac{v_{i-1}}{v_i}\right) a_{i-1}^p + \left(1 - \frac{v_{i-1}}{v_i}\right) a_n^p \end{aligned}$$

□

It is instructive to pause here to reconsider the last lemma from a slightly different point of view that will enable us to gain some insight into the more general

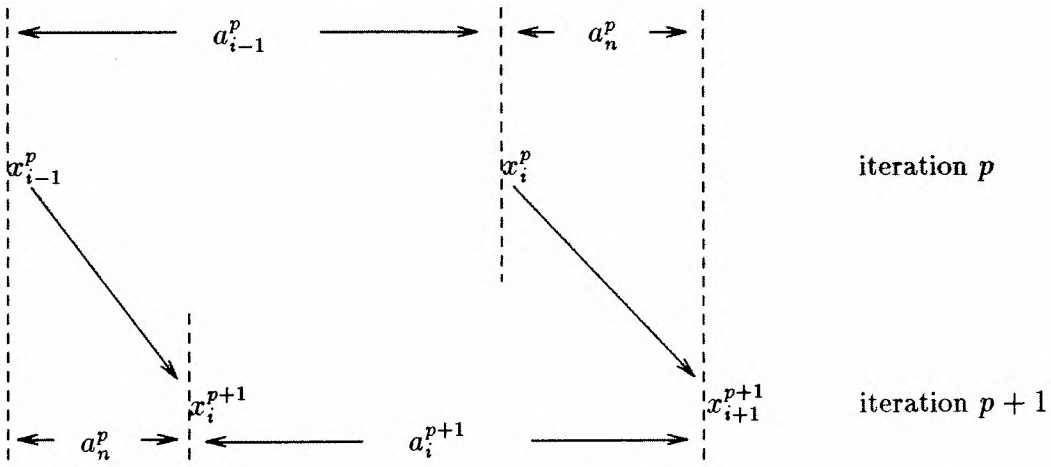


Figure 5.1: Workers $i-1$ and i at iteration p form the allocation for worker i at iteration $p+1$.

accelerating model presented in Chapter 3. The equality

$$a_i^{p+1} = (v_{i-1}/v_i)a_{i-1}^p + (1 - v_{i-1}/v_i)a_n^p$$

can be derived as shown in Figure 5.1 where we show the case where $x_i^{p+1} < x_i^p$ (the other case follows similarly). The allocation a_i^{p+1} can be written as

$$a_i^{p+1} = a_n^p + (a_{i-1}^p - a_n^p)\frac{t_i\left(x_i^{p+1}, x_i^p\right)}{t_{i-1}\left(x_i^{p+1}, x_i^p\right)}.$$

Where the term $(a_{i-1}^p - a_n^p)$, the time for worker $i-1$ to process the interval $[x_i^{p+1}, x_i^p]$, is converted by the ratio $t_i\left(x_i^{p+1}, x_i^p\right)/t_{i-1}\left(x_i^{p+1}, x_i^p\right)$ to the time for worker i to cover this interval. For the uniform accelerating model this ratio is always v_{i-1}/v_i, regardless of the interval, and our derivation coincides with Lemma 5.2. In the general accelerating model the value of such a ratio depends on the particular interval in question, but we know such a ratio is always less than one. Thus one can write each allocation at each iteration of a general non-blocking accelerating system as a strict convex combination of two other allocations. Then in an informal sense, the

maximum allocation decreases and the system tends toward a fixed point. As the system gets closer to the fixed point, the interval for which each ratio is derived is also converging. Therefore each ratio is converging, and the transition matrix converges to a constant. When we account for blocking the proof becomes more complex, but the underlying reason for convergence of an accelerating system is that the ratio of the time required for worker $i+1$ to the time for worker i to complete any interval is strictly less than one. One should note that when this ratio equals one all workers are identical, and the convergence to a fixed point fails as we will see in Section 5.2. When this ratio exceeds one the behavior is complex and unpredictable as we saw in Chapter 4.

Thus from Lemma 5.2, unlike the general accelerating system presented in Chapter 3, we can write $\mathbf{a}^{p+1} = T\mathbf{a}^p$ where T is unchanged from iteration to iteration:

$$T = \begin{bmatrix} 0 & 0 & 0 & \dots & & 0 & 1 \\ v_1/v_2 & 0 & 0 & \dots & & 0 & 1 - v_1/v_2 \\ 0 & v_2/v_3 & 0 & \dots & & 0 & 1 - v_2/v_3 \\ & \ddots & \ddots & \ddots & & \vdots & \vdots \\ 0 & 0 & \dots & 0 & v_{n-2}/v_{n-1} & 0 & 1 - v_{n-2}/v_{n-1} \\ 0 & 0 & \dots & 0 & 0 & v_{n-1}/v_n & 1 - v_{n-1}/v_n \end{bmatrix}.$$

We can interpret T as the transition matrix of an n-state Markov chain. Since no two workers have the same speed factor, each row except the first has exactly two positive elements. Once in state 1 a transition to state n is guaranteed, while

in any state $i > 1$ a transition to state $i-1$ occurs with probability v_{i-1}/v_i and a transition to state n occurs with probability $1 - v_{i-1}/v_i$. Therefore, any state can be reached from any other, and so the Markov chain is irreducible. Furthermore, the Markov chain is aperiodic since state n has a transition to itself of probability $1 - v_{n-1}/v_n$.

Any finite Markov chain that is irreducible and aperiodic has limiting probabilities, π_j $(j = 1, \ldots, n)$, such that each row of T^∞ is $(\pi_1, \pi_2, \ldots, \pi_n)$. Furthermore, the π_j are the unique solution to the equations

$$\pi T^{-1} = \pi \quad \text{and} \quad \mathbf{1}\pi = 1. \tag{5.1}$$

Solving for π yields

$$\pi_j = \frac{v_j}{\sum_{i=1}^n v_i} \quad \text{for } j = 1, \ldots, n.$$

Thus the allocations converge to $\mathbf{a}^* = T^\infty \mathbf{a}^0$ where each allocation

$$\begin{aligned} a_i^* &= \frac{\sum_{j=1}^n a_j^0 v_j}{\sum_{j=1}^n v_j} \\ &= \frac{\sum_{j=1}^n (x_{j+1}^0 - x_j^0)}{\sum_{j=1}^n v_j} \\ &= \frac{1}{\sum_{j=1}^n v_j}. \end{aligned}$$

Thus each worker i repeatedly executes an interval of work content of width $v_i / \sum_{j=1}^n v_j$, so that the interval for worker i is

$$\left[\frac{\sum_{j=1}^{i-1} v_j}{\sum_{j=1}^n v_j}, \frac{\sum_{j=1}^{i} v_j}{\sum_{j=1}^n v_j} \right),$$

and therefore

$$x_i^* = \frac{\sum_{j=1}^{i-1} v_j}{\sum_{j=1}^n v_j}. \tag{5.2}$$

The production rate is then $\sum_{j=1}^{n} v_j$, the largest possible. This is expected since workers are never delayed and their uniform speed means that it is irrelevant exactly where on the production line they produce.

When the skills of the workers approximate a uniform model, a manager now has some guidelines in planning the production line. For a given set of workers the manager can use equation 5.2 to estimate a non-blocking fixed point for the line. One must now check if such a fixed point is feasible; that is, that indeed no blocking will occur. This is done by insuring that each worker completes her initial work station before her predecessor requires its use. If the fixed point is feasible (with some slack) the manager may consider adding another worker to the line or invest in additional training to raise the skills of the workers. If, however, the non-blocking fixed point is not feasible, then the line as it is configured is overstaffed. One or more workers may need to be removed or possibly a station of large processing requirement can be broken into multiple tasks or a machine added in parallel. If the precedence constraints on the station ordering allow, one may be able to remove the blocking by a reordering of the work stations. (There is typically some leeway in station ordering for the apparel industry.) The next section will show the best way to order stations for this model.

5.1.2 ordering of work stations

Alternate ordering of work stations does not affect the production rate if both yield a fixed point with no blocking — it's as large as possible. However, we are concerned with a reordering of the work stations that might move a system from a fixed point with blocking to one of improved production rate.

We noted in Chapter 3 that for the general accelerating model an ordering of the

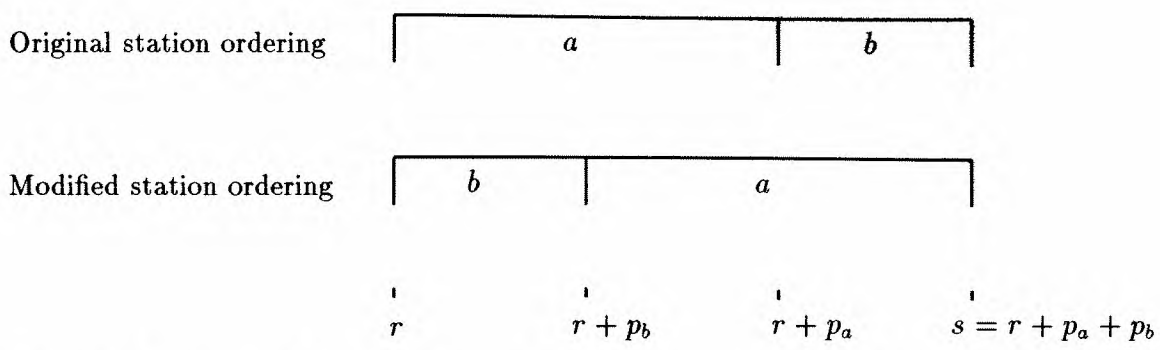


Figure 5.2: A pair of neighboring work stations a and b, with $p_a > p_b$, are interchanged to move toward a nondecreasing order of the processing requirements of the work stations.

stations based on the p_js independent of any restrictions on the velocity functions is meaningless. On the other hand, such an ordering of the stations for the uniform accelerating model is meaningful because we have stipulated that each velocity function be held constant. Thus one is not free to rescale the processing times without destroying the constant velocity functions. Therefore we use the uniform model to help gain some insight on how to best sequence the work stations.

Theorem 5.3 *For a uniform accelerating system, an ordering of the work stations such that* $p_1 \leq p_2 \leq \cdots \leq p_m$ *achieves as large a production rate as any other ordering of stations.*

Proof We consider an original ordering of the stations that is not in nondecreasing order of the processing requirements and let $\mathbf{x}$ be the fixed point in position space and $\mathbf{a}$ the corresponding fixed point in allocation space. We show that if we exchange any two adjacent stations a and b, with $p_a > p_b$ as shown in Figure 5.2, then we can construct vectors $\mathbf{x}'$ and $\mathbf{a}'$ such that each $a'_{\max} \leq a_{\max}$. Therefore, since the max allocation never increases the new line must converge to a fixed point of no greater cycle time than $a_{\max}$. Thus after repeating such interchanges the theorem holds.

We consider two cases.

1. There is not a worker i such that $x_i \in (r, r + p_a)$.
   We let $\mathbf{x}' = \mathbf{x}$ and k be the largest index such that $x_k < r$.

   (a) $x_{k+1} > s$
   Since no worker positions are within the interval (r,s) we have $\mathbf{a}' = \mathbf{a}$.

   (b) $x_{k+1} \in (r + p_a, s)$
   There cannot be multiple workers at $r + p_a$ since such workers would have delay or inherited allocations of value less than a_k. Thus we only need consider one worker $x_{k+1} \in (r + p_a, s)$.
   Both a_k and a'_k are simple allocations since $v_k < v_{k+1}$ and therefore $\mathbf{a}' = \mathbf{a}$.

2. There is a worker i such that $x_i \in (r, r + p_a)$

   (a) $x_{i+1} \in (r + p_a, s)$
   We let $x'_k = x_k$ for all $k \neq i$, and $x'_i = \min\{x_i, r + p_b\}$.
   We first bound a'_i. If a'_i is simple then $x'_i = x_i$ and thus $a'_i = a_i$. Otherwise a'_i is the time required for the combined efforts of workers i and $i + 1$ to complete station a, which must be less than a_{i-1}, which is at least the time required for the combined efforts of workers $i - 1$ and i to complete station a.
   We now bound a'_{i-1}. If $x'_i < r + p_b$ then $a'_{i-1} \leq a_{i-1}$ since worker i will finish her current station in the modified line before she will in the original line. If $x'_i = r + p_b$ then $a'_{i-1} = \max[(x'_i - x'_{i-1})/v_{i-1}, a'_i] \leq a_{i-1}$.

   (b) $x_{i+1} > s$
   We let $\mathbf{x}' = \mathbf{x}$.
   Therefore $a'_i = a_i$.
   We now bound a'_{i-1}. Suppose $x'_i \geq r + p_b$. If a'_{i-1} is simple so is a_{i-1} and thus $a'_{i-1} = a_{i-1}$. If a'_{i-1} is a delay allocation then it is the time for $i - 1$

and i to combine efforts on station a. a_{i-1} is at least the time required for $i-1$ and i to combine efforts on station a and furthermore x_i is closer to the end of station a than x'_i and therefore $a_{i-1} > a'_{i-1}$. Suppose $x'_i < r + p_b$, then $a'_{i-1} < a_{i-1}$ since worker i will finish her current station in the modified line before she will in the original line.

□

The sequencing of stations from smallest to largest processing times for an accelerating system is intuitive; the fastest workers are assigned the stations of largest processing requirements. Also consider the allocation equations: The processing requirement for every station is either contained in the allocation for a single worker (a simple allocation), or shared by adjacent workers (a delay allocation). Thus if slow workers are processing on a station of large processing time their allocation must reflect their processing time on that station. This leads to the following conjecture whose proof seems difficult since we are unable to characterize the production rate of non-accelerating systems.

Conjecture 5.1 *Consider a line where each worker has a distinct constant velocity. The system where workers are ordered from slowest to fastest and work stations are ordered such that* $p_1 \leq p_2 \leq \cdots \leq p_m$ *achieves as large a production rate as any other ordering of workers and stations.*

5.1.3 production rate

We saw in Chapter 3 that an accelerated sequence of workers can be worse by a factor n from an alternate sequence. However, this bound can be substantially improved for the uniform model.

Theorem 5.4 *Consider a set of workers of distinct constant velocities. An accelerated ordering of the workers is within a factor* $(n+3)/4$ *of the best achievable by any other sequence of workers.*

Proof Let $1, \ldots, n$ be a uniform accelerated ordering or workers with v_n normalized to 1. An accelerated line always keeps the fastest worker busy and therefore the worst production rate is realized when at the fixed point worker n is forced to produce as large a proportion of the line as possible for a given line and set of workers. Figure 5.3 illustrates such worst case fixed points. The fastest worker processes on all stations, with help only from worker $n-1$ on the portion βp_1, $0 < \beta < 1$, of the first station. The rest of the stations have small processing requirements. The cycle time of the line is $1 - \beta p_1$, and thus we have that the worst case production rate, $z_w = (1 - \beta p_1)^{-1}$. The best possible production rate, $z_b = \min\{\sum v_i, 1/p_1\}$. So we look for an upper bound to

$$\frac{z_b}{z_w} = \min\left\{\sum v_i(1-\beta p_1), \frac{1}{p_1}(1-\beta p_1)\right\}.$$

The first term holds when $p \leq 1/\sum v_i$, while the second term holds when $p \geq 1/\sum v_i$. The term $\sum v_i(1-\beta p_1)$ increases with p_1 and thus is maximized for $p = 1/\sum v_i$. For the second term, $1/p_1(1-\beta p_1)$, the first component decreases with p_1 while the second increases with p_1. But if we consider increasing p_1 by multiplying by any $\gamma > 1$ and considering their difference, we get

$$\frac{1-\beta p_1}{p_1} - \frac{1-\beta\gamma p_1}{\gamma p_1} = \frac{1-1/\gamma}{p_1} > 0.$$

Therefore the second term is also maximized when p_1 is at its lower bound of $1/\sum v_i$. We therefore can consider either term in our analysis by setting $p_1 = 1/\sum v_i$.

We now evaluate $1/p_1(1-\beta p_1)$. Referring to Figure 5.3, the time for worker n to complete $1-p$ equals the time for worker $n-1$ to process βp_1, so we have

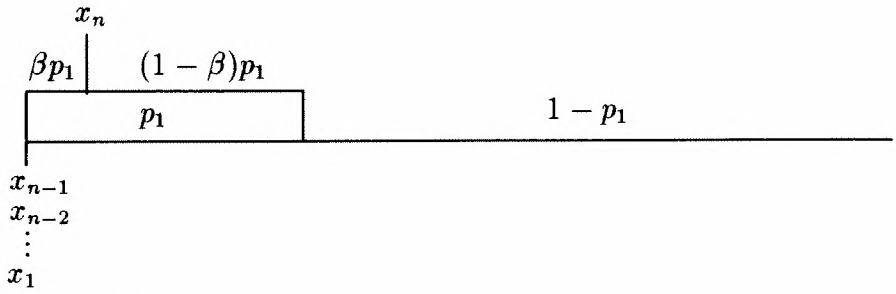


Figure 5.3: The fixed point of an accelerating line, where the fastest worker processes on all stations and workers $1, \ldots, n-2$ are idle.

$1-p_1 = \beta p_1 / v_{n-1}$ and thus the cycle time is $1-\beta p_1 = 1-(1-p_1)v_{n-1}$. Furthermore, in the accelerated line we do not use workers $1, \ldots, n-2$, and the production rate is maximized for the best case line when $v_1, \ldots, v_{n-2}$ is maximized. Thus we form an upper bound on z_b/z_w by setting each $v_i = v_{n-1}$ for all $i < n-1$, and so we let $k = \sum v_i = (n-1)v_{n-1} + 1$ and evaluate

$$\begin{aligned} \frac{z_b}{z_w} &< \frac{1}{p_1}(1-\beta p_1) \\ &= k\left(1-\left(1-\frac{1}{k}\right)v_{n-1}\right) \\ &= k-(k-1)v_{n-1} \\ &= (n-1)v_{n-1}(1-v_{n-1})+1. \end{aligned}$$

Since $0 < v_{n-1} < 1$, the term $v_{n-1}(1-v_{n-1})$ is maximized at $1/4$, and thus

$$\frac{z_b}{z_w} < \frac{n-1}{4} + 1 = \frac{n+3}{4}.$$

□

The bound of Theorem 5.4 is tight. Consider the line where, as in Figure 5.3, $p_1 = 1/2$ and $\mathbf{v} = (1/2-\epsilon, 1/2, 1)$. The fixed point for the accelerating line is $\mathbf{x} = (0, 0, 1/4)$ and yields a production rate of $4/3$. The best possible production

rate is $\sum v_i = 2 - \epsilon$, and thus as ϵ tends to 0 the bound $z_b/z_w \to 3/2$ which is the bound derived in Theorem 5.4.

5.2 Identical worker model

A simplified model of the TSS line when all workers are of essentially the same skill level can be useful in estimating the production rate or the appropriate number of workers for a TSS line. In general, such a model is useful when the time to complete a task is independent of the worker.

When all workers are the same skill level, then, since the work standard can be rescaled, we can assume without loss of generality that all $v_i \approx 1$. We again make the modelling assumption that the time to walk back and preëmpt another worker is small.

For this simple model the maximum production rate is easy to express. Since each worker can produce no more than 1 item per time unit the production rate cannot exceed n. In addition, no matter how many workers are on the line, the production rate cannot exceed $1/p_{\max}$, where $p_{\max}$ is the largest processing requirement. Thus the maximum production rate of n workers is $\min\{n, 1/p_{\max}\}$ items per time unit. We show that the TSS line will converge to achieve this maximum production rate on average over each n consecutive items.

To help analyze the dynamics of the TSS line we introduce a simpler production line we call the Forward line, and show that this line is functionally equivalent to the TSS line when all workers are identical. In the Forward line each worker is again devoted to a single item, but instead of walking back to preëmpt your predecessor when an item completes production, workers just circle the production line moving from station n to station 1 to retrieve a new item from the input buffer. This behavior is realized by requiring each worker to independently follow this rule:

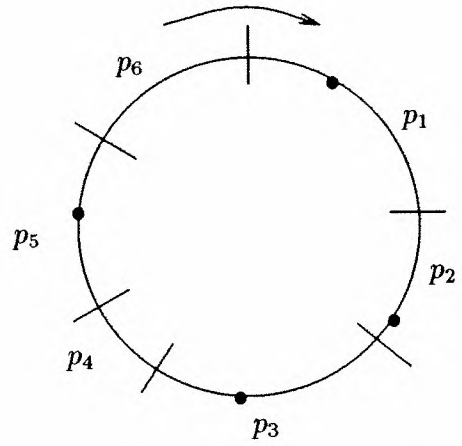


Figure 5.4: Item movement with identical workers on a TSS line is equivalent to the Forward line. Here 4 workers (•) move around a cyclic production line with 6 work stations.

Forward Rule Remain devoted to a single item, and process it on successive work stations, queueing before a busy work station if necessary. When you complete processing your item circle back to the first station and retrieve a new item.

Thus each worker on a Forward line operates all work stations.

Workers of a Forward line can be viewed as moving at a constant rate around a production circle unless they are blocked by a busy work station, see Figure 5.4.

Lemma 5.5 *The movement of items in the Forward line is identical to that of the TSS line when workers are identical.*

Proof When a unit completes production in the TSS line each worker $2, \ldots, n$ replaces her predecessor and worker 1 retrieves a new item from the input buffer. Thus at each reset, all items remain at the same location except that a just completed item is replaced by a new one entering the system. But this is just the behavior of the Forward line, and since all workers are identical the movement of items in the Forward and TSS lines is indistinguishable. □

We can now analyze the dynamics of the identical worker model by either a Forward line or a TSS line perspective, whichever is most convenient.

Workers are blocked at a work station if they arrive "too close" to each other, but will then be "separated" as they complete processing on the station. The next two observations formalize these notions about the Forward line.

Observation 5.1 *The rate at which workers leave station j is no more than $1/p_j$.*

Observation 5.2 *Consider subsequent workers that arrive at an idle work station j. The second worker will be blocked at station j only if their interarrival time is less than p_j.*

The next lemma shows that a queue can develop only finitely often at most work stations.

Lemma 5.6 *After finite time, workers on a Forward line can only be blocked at a work station of largest processing time.*

Proof For convenience we renumber the work stations (in the natural order as workers circle the production line), such that a work station with processing requirement $p_{\max}$ is numbered 1. Now suppose on the contrary that work station j is a station of smallest index with $p_j < p_{\max}$ such that workers are blocked at station j infinitely often. But from Observation 5.1, workers leave station 1 at a rate no greater than $1/p_{\max}$, and any queue at station j empties at a rate of $1/p_j > 1/p_{\max}$. Thus after finite time the queue at station j must empty, and from Observation 5.2, since a queue can reform at station j only if workers arrive at a rate faster than

We can now analyze the dynamics of the identical worker model by either a Forward line or a TSS line perspective, whichever is most convenient.

Workers are blocked at a work station if they arrive "too close" to each other, but will then be "separated" as they complete processing on the station. The next two observations formalize these notions about the Forward line.

Observation 5.1 *The rate at which workers leave station j is no more than $1/p_j$.*

Observation 5.2 *Consider subsequent workers that arrive at an idle work station j. The second worker will be blocked at station j only if their interarrival time is less than p_j.*

The next lemma shows that a queue can develop only finitely often at most work stations.

Lemma 5.6 *After finite time, workers on a Forward line can only be blocked at a work station of largest processing time.*

Proof For convenience we renumber the work stations (in the natural order as workers circle the production line), such that a work station with processing requirement $p_{\max}$ is numbered 1. Now suppose on the contrary that work station j is a station of smallest index with $p_j < p_{\max}$ such that workers are blocked at station j infinitely often. But from Observation 5.1, workers leave station 1 at a rate no greater than $1/p_{\max}$, and any queue at station j empties at a rate of $1/p_j > 1/p_{\max}$. Thus after finite time the queue at station j must empty, and from Observation 5.2, since a queue can reform at station j only if workers arrive at a rate faster than

$1/p_j$, we must conclude no such work station j can queue workers infinitely often. □

Thus from Lemma 5.6, after finite time, workers can only be blocked at a work station of processing requirement $p_{\max}$. If more than one station has processing requirement $p_{\max}$, then multiple queues may persist, however, the following lemma gives a condition for even these queues to last indefinitely.

Lemma 5.7 *Let $p_j = p_{\max}$, then after some finite time, if a worker is not waiting for station j when it becomes free then no worker will ever again be blocked at station j.*

Proof From Lemma 5.6, after finite time, workers arrive at station j at a rate no faster than $1/p_{\max}$, thus from Observation 5.2, once workers cease being blocked at station j, queueing cannot recur. □

Theorem 5.8 *The TSS line with identical workers converges to the maximum production rate of* $\min\{n, 1/p_{\max}\}$.

Proof From Lemma 5.6, after finite time workers can only queue at a station with maximum processing requirement, and from Lemma 5.7, such queueing will either cease or persist forever. If it persists forever, then we achieve the maximum production rate of $1/p_{\max}$. If it ceases then all workers stay busy and we achieve a maximum production rate of n. □

From Theorem 5.8 the following observation holds.

Observation 5.3 *The ordering of the work stations does not affect the production rate of the TSS line when workers are identical.*

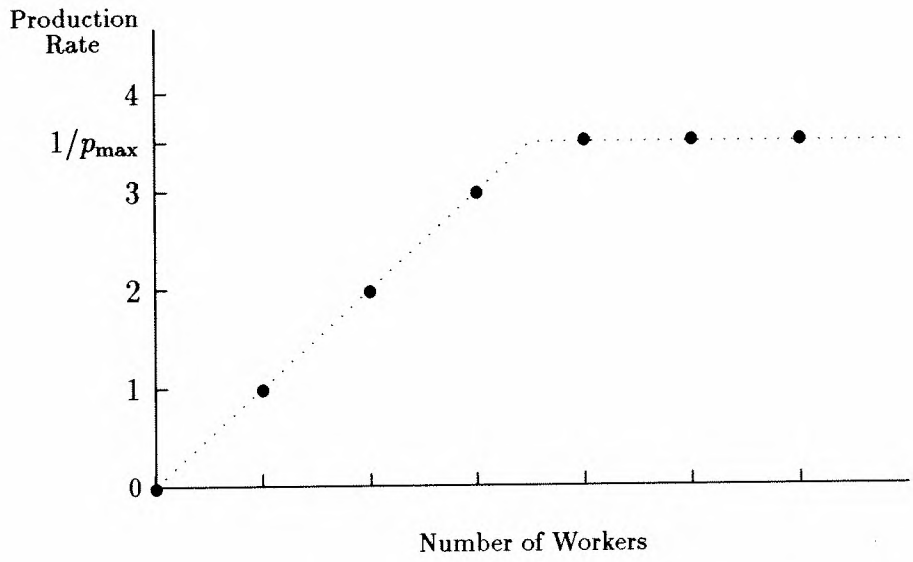


Figure 5.5: Production rate increases linearly in the number of workers n until $n \geq 1/p_{\max}$.

The expression $\min\{n, 1/p_{\max}\}$ for maximum production rate achieved by the TSS line with identical workers makes clear the effect of adding workers to the line: both production rate and station utilization increase proportionally with n until $n \geq 1/p_{\max}$, at which point the line becomes overstaffed and they increase no further, see Figure 5.5. When $n \geq 1/p_{\max}$ the production rate is throttled by a work station of processing requirement $p_{\max}$ and we call such a station a *bottleneck* station. One should note that a line will have a bottleneck only when n becomes too large, and conversely, there is always a sufficiently large n to induce a bottleneck station in any line. We will refer to an overstaffed line as having multiple bottleneck stations when $p_j = p_{\max}$ for more than one j, but one should note that removal or reduction in the processing requirement in just one of these bottleneck stations will not improve the production rate.

We saw in Chapter 3 that an accelerating TSS line converged to a fixed point

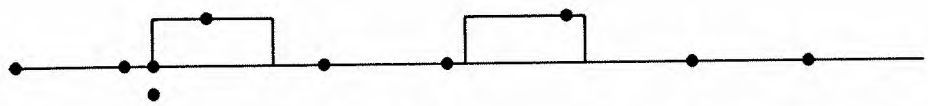


Figure 5.6: We show an overstaffed line at reset with two bottleneck stations. All workers are spaced apart by $p_{\max}$ until they possibly queue at a bottleneck. Here they only queue at the first bottleneck station. Note that the allocation of each worker is $p_{\max}$, whether its simple, delay or inherited, and thus the line is at a fixed point.

while non-accelerating lines tended to converge to a unique limit cycle if workers are not started at a fixed point. The limit cycles for non-accelerating lines were unpredictable and the cycles could have extraordinary length. The next lemma shows that a TSS line with identical workers always converges to a fixed point or a limit cycle of bounded length.

Lemma 5.9 *Any TSS line with identical workers converges to a fixed point or limit cycle of length no greater than n.*

Proof From Lemmata 5.6 and 5.7, after finite time, queues either persist at one or more bottleneck stations or workers remain busy. If queues persist at one or more bottleneck stations then workers leave each bottleneck with a work allocation of exactly $p_{\max}$ and cannot be blocked until they reach another bottleneck station (from a Forward line perspective), at which time they may queue, but their allocation will then be an inherited or delay allocation of value $p_{\max}$, see Figure 5.6. Thus at each reset of the line (from a TSS perspective) all allocations are the same and therefore, from Theorem 2.2 the line is at a fixed point.

If in finite time the workers are never idled, then $a_1^{p+1} = a_n^p$ and $a_i^{p+1} = a_{i-1}^p$, for $i > 1$, and thus each worker i will repeat the same allocation at least every n iterations. Therefore, if a bottleneck is present, the line converges to a single fixed

point, otherwise, it converges to a limit cycle of length no greater than n. □

While the limit cycle of a TSS line with identical workers is in some sense simpler, since it is bounded in length by n, the following observation adds some complexity to this system we have not observed when workers are of distinct skill levels.

Observation 5.4 *For a TSS line with n identical workers the number of distinct limit cycles of length greater than one can be infinite.*

This observation follows by considering a line with two workers and stations of sufficiently small processing requirement that blocking does not occur. Then if the workers are started at position $(0, x_2)$ they will cycle between $(0, x_2)$ and $(0, 1 - x_2)$ for all $x_2 \in (0, 1)$. Only blocking at work stations of large processing requirement can prevent the possibility of an infinite number of limit cycles. In this two worker example, if two stations of processing requirement $1/2$ each are considered, the only limit cycle is of length one at the fixed point $(1/2, 1/2)$ of worker positions.

The line achieves, on average over n iterations, the maximum production rate no matter which limit cycle the system is trapped. However, from a management perspective, some limit cycles are more desirable than others. This is because the particular limit cycle will determine the interval of work content each worker must process; the smaller the interval for each worker the less training is required. For instance, consider again the example of two workers with limit cycle $(0, x_2)$ and $(0, 1 - x_2)$. If x_2 is close to $1/2$ each worker will repeat an interval of work content of width close to $1/2$, and thus have minimal training. If however x_2 is close to 1, each worker must be trained on nearly all the stations on the line. Thus in general the training is minimized when all allocations are equal (at a fixed point) and the training increases as allocations differ is value.

CHAPTER 6

Related work

There has been a wealth of work on production lines but most differs from ours in not modelling the worker explicitly. Generally the worker is simply identified with a fixed work station. Our model seems unusual in that we treat workers as resources distinct from the stations at which they might work; moreover, we model even individual workers by specifying their skill levels.

Ostolaza, McClain, and Thomas (1991) described assembly lines with some additional flexibility: For each pair of adjacent work stations j and $j+1$ there is a "shared task" that may be done at either station, and that decision can be made during production. Ostolaza et al. gave evidence that, if all processing times are exponentially distributed, then the line can function effectively even if each station uses a simple rule to decide which of its waiting tasks to perform next and whether to perform the shared task itself or to pass responsibility for it to the following station. Their model was intended to be suggestive, since the distribution of task times is chosen for tractability rather than plausibility.

The production line described by Ostolaza et al. differs in several ways from the TSS line. First, their workers are identified with work stations and the line

is balanced by clever management of work-in-process inventory. In contrast, the workers on a TSS line move to where the work is and so there is no work-in-process inventory beyond that in the hands of the workers. Another difference is that the line of Ostolaza et al. does not allow tasks to be split between work stations (workers), while the TSS line allows task preëmption at any time. This suggests that the line of Ostolaza et al. might be more appropriate where task preëmption is very costly and it is not too expensive to provide two sets of tools for each of the shared tasks. A TSS line seems more appropriate when preëmption is not costly and tools are expensive, as in the apparel industry.

Schroer, et al. built a simulation model of a particular TSS line they observed at a trade show (Schroer, Wang, and Ziemke, 1991). Because the point of their work was to demonstrate capabilities in object-oriented simulation, they did not pursue analysis of the TSS system. Instead they were content to gather statistics on the single instance they simulated. They reached no general conclusions about TSS lines. In contrast, our model explains the dynamics of the system, so that its behavior can be understood rather than simply replicated.

In building their simulation Schroer, et al. assumed, as did we, that the time for a worker to move between stations is negligible. Their formulation also differed in several ways from ours. For example, they only considered an identical worker model. They also injected a stochastic element by assuming that the processing time at each station j is normally distributed about its mean μ_j, with standard deviation $0.15\mu_j$; however, this deviation is sufficiently small compared to the allocations of the workers that significant blocking is unlikely. Therefore, since the mean work content at each station remains stationary, their system is expected to operate like our deterministic model with $p_j = \mu_j$ (and apparently did, since our analytic model

can predict all of their statistics).

CHAPTER 7

Other issues

7.1 Preëmption costs

Although preëmptions for the apparel industry are surprisingly efficient, not all production lines can tolerate workers frequently taking over an item of another worker. There is a couple of ways to reduce the costs of such preëmptions. One way is to associate a batch of k items with each worker instead of a single item. Thus each worker would process k items on a station before moving on to the next. On reset, each worker preëmpts just the item in progress of her predecessor, but takes over the entire batch of k items. Each batch is preëmpted exactly $n-1$ times before being completed, and thus the cost of preëmpting is now amortized over k items. The savings in preëmption costs must be weighed against an increased cost of material handling and work-in-process inventory.

Another variant of the TSS rule eliminates preëmptions completely. In this rule O(n) buffer space is used so that on reset workers retrieve items from a buffer instead of preëmpting a predecessor. Simulations of such a rule are encouraging, but mathematical analysis of the system is not yet complete.

7.2 Other topologies

An interesting and practical extension is to allow the line to have identical parallel stations. For example, consider a topology with k_j copies of station j. Then we may extend the TSS rule as follows: Sequence the workers from slowest to fastest and identify each worker by her number in the sequence. Now we stipulate that worker i can preëmpt only worker $i-1$. (This ensures that a slower worker can never be closer to the end of the line than a faster worker.) The corresponding change to our model is to allow up to k_j x_i's to have values simultaneously within the interval of work corresponding to station j. We know an accelerating system converges when $k_j = 1$ and when $k_j = n$ for each station j. From simulations we observe such lines converging for any values of k_j, but the proof of convergence is not yet complete.

Another topological extension is to allow tasks which only require the worker to set up the item at the station and then retrieve it at a later time. A steam press is such an example in the apparel industry where a worker will set the garment to be pressed and then may perform other tasks before retrieving the pressed garment. One might extend the TSS rule and analysis to include such tasks.

Another interesting topological extension is to allow for more general networks of material flow such as TSS lines that feed into other TSS lines. How should workers move on such flow arborescences? Should workers move between connecting TSS lines? Must buffers be used at intersection points?

7.3 Other decentralized rules

The TSS rule can be considered decentralized since each worker follows her own rule without any management (centralized) authority. We look at two variants of the

TSS rule that seem promising but turn out to be inferior.

The Forward rule is a simple variant described in Chapter 5 that has each worker circle from station n back to station 1 instead of preëmpting her predecessor. While such a rule does eliminate preëmptions it has a couple of drawbacks. For one, each worker must be trained on all tasks. Second, if workers are of different skill levels then a faster worker will eventually catch a slower one, and the production rate of each worker will drop to that of the slowest worker. One is not able to preëmpt a worker from behind efficiently since there is still multiple workers requiring the same station — someone must wait. The Forward rule would seem viable only if preëmption costs were very high or tasks were so similar and independent of the operator that workers can master them all and perform them at approximately the same rate.

In another variant of the TSS rule, a worker who finds a station busy will leave their item in a buffer and begin to follow the backward part of the TSS rule instead of waiting for the station to become available. The idea here is to keep a worker busy and work off the buffers at a later time. The problem is that buffers tend to build in front of stations of large processing times and workers fail to migrate down the production line to work off these buffers later — instead the workers remain bunched-up building work-in-process inventory. Another way to view the problem with such an approach is to consider a bottleneck station and thus the effective production rate of the line is dependent on keeping the bottleneck station busy. If workers do not queue for the bottleneck station then no one will be waiting when the station becomes free and therefore the bottleneck station experiences idle time and the production rate must suffer.

CHAPTER 8

Conclusions

In our model an accelerating TSS line has many attractive properties. The TSS rule is *simple*, which makes it easy for the workers to learn. It is *parsimonious* in its data requirements, which are only the relative speeds of the workers (not even their values); and it does not require knowledge of task times. The TSS line is *adaptive*: The line configures itself without management intervention. Also, the TSS line has negligible work-in-process inventory: one item for each worker. The TSS line is *easy to manage*: The production rate can be fine-tuned by adjusting the number of workers; and because the line does not exhibit anomalous behavior, adding workers never reduces the production rate and removing workers never increases it.

A TSS line can be implemented "on top of" a classical assembly line: First the tasks are distributed over the stations, which will typically result in an imperfect allocation of work among stations. This first allocation is static and unchanging. Then workers, sequenced from slowest to fastest, follow the TSS rule on the line. A second allocation of work emerges, this time among the workers. This second allocation is dynamic and self-correcting and it can smooth over imperfections in the underlying static allocation.

We proved self-balancing behavior of accelerating TSS lines for a very general velocity function. However, since the function was so general, we could not derive simple expressions to characterize the fixed point, nor could we provide simple rules for how best to sequence the work stations. Furthermore, such a general function led to pathological examples of poor performance by an accelerated sequence of workers. We then considered a restricted accelerated system, the uniform model, that was still powerful enough to model actual production lines (it's a good model for the apparel industry). Such a model provided a simple expression for a non-blocking fixed point and therefore a predictor of the production rate. We were able to show that such an accelerated system performs at its best when the work stations are sequenced from smallest to largest processing requirements. We were further able to strengthen the bound on the performance of an accelerated line verses any other sequence of workers. Thus the uniform model provided insight into the design and performance of TSS lines.

We have shown that, if the workers are sequenced from slowest to fastest, then a TSS system is robust in the sense that its long term behavior is independent of the starting positions of the workers. There are at least two other senses in which the robustness of TSS might be studied. One is to determine whether, if an actual production line is close to our idealized model, then observed behavior will be close to predicted behavior. If so, then it might not matter if details of our model are only approximate, as long as they are not egregiously false. (This type of robustness is known in dynamical systems theory as "structural stability".)

Another type of robustness is the resistance to stochastic perturbations. For example, one might wonder about the effect of allowing the processing time at a station to include an element of randomness. We have not worked out the details of

this, but they seem predictable: that for suitable distributions of the task times an accelerating TSS line will converge not to a fixed point, but to a random variable, the distribution of which is independent of the starting positions of the workers. Furthermore, this random variable will have small variance if the variance of task times is small.

It might also be of mathematical interest to catalogue and explain the apparently complicated behavior possible when workers are sequenced other than slowest to fastest. We do not fully understand the complexity of such a system. We have observed hundreds of simulations and all appeared to converge to limit cycles, but we do not know whether this behavior is universal.

Finally, suppose a manager is faced with a given set of workers and tasks and that an accelerating system does not appear easy to form. As we discussed in Chapter 3 it is unimportant how some workers perform on some of the stations. The manager must try to find an ordering of stations and workers such that the line behaves as an accelerating line while also achieving a high production rate.

Bibliography

[1] B. Bollobás. *Linear Analysis*, Cambridge University Press (1990).

[2] R. L. Devaney. *An Introduction to Chaotic Dynamical Systems*, 2nd ed., Addison-Wesley, Reading, Mass. (1989).

[3] F. Morrison. *The Art of Modelling Dynamic Systems: Forecasting for Chaos, Randomness, and Determinism*, John Wiley & Sons, New York (1991).

[4] J. Ostolaza, J. McClain, and J. Thomas. "The use of dynamic (state-dependent) assembly line balancing to improve throughput", *J. Mfg. Oper. Mgt.* **3**:105–133 (1990).

[5] B. J. Schroer, J. Wang, and M. C. Ziemke. "A look at TSS through simulation", *Bobbin* magazine, July, 114–119 (1991).

[6] T. Yoshida, H. Mori, and H. Shigematsu. "Analytic study of chaos of the tent map: band structures, power spectra, and critical behaviors", *Journal of Statistical Physics* **31**(2):279–308 (1983).

Vita

Donald David Eisenstein was born in Dallas, Texas on June 23, 1961. He graduated from Lake Highlands High School in Dallas in 1979. He then studied at Southern Methodist University where he received Bachelor degrees in Engineering Management and Mathematical Science in 1982. He then moved to Atlanta, Georgia to attend the Georgia Institute of Technology where he received his Master of Science in Operations Research in 1983 from the School of Industrial and Systems Engineering. He then worked as an engineer at E-Systems for four years before returning to the School of Industrial and Systems Engineering at Georgia Tech where he received his Doctor of Philosophy in 1992.